Universidad Carlos III de Madrid

Escuela Politécnica Superior
Departamento de Informática

Trabajo de Fin de Grado

DISEÑO E IMPLEMENTACIÓN DE UNA PLATAFORMA AUTÓNOMA PARA EL CONTROL DE SISTEMAS SERVOMECÁNICOS

AUTOR: Fernando Gallego Hernández

TUTOR: Javier Fernández Muñoz
COTUTOR: Alberto Jardón Huete

Fecha: 25 de junio de 2013

A mis padres,
muchas gracias por todo.

"Si vives cada día de tu vida como si fuera el último, algún día realmente tendrás razón."

Steve Jobs. Discurso de Stanford (2005).

AGRADECIMIENTOS

Con la realización de este trabajo concluyen 4 años desde que comencé a estudiar el grado en Ingeniería Informática. Corresponde con una etapa de esfuerzo personal, capacidad de superación, adquisición de conocimientos y disfrute por haber podido estudiar algo que siempre me ha apasionado.

En primer lugar, dar las gracias a mis padres y mis hermanas, ya que sin su apoyo constante no hubiera podido llegar hasta aquí.

En segundo, a todos los profesores de la facultad de los que he podido aprender mucho y sufrir en algún examen con alguno, también sea dicho. Especialmente quiero dar las gracias a Javier Muñoz, tutor de mi trabajo de fin de grado, y además profesor de la asignatura de 4º de Sistemas en Tiempo Real, una de las que más he disfrutado de toda la carrara.

En tercero, a los miembros del Servicio de Informática de Comenarejo y a Juanto en especial, por haberme aguantado durante estos últimos 3 años y por haber podido trabajar con ellos como becario en el departamento.

Para finalizar, no puedo olvidarme de todos mis amigos que han estado ahí en los malos y buenos momentos, dándome siempre ánimos para no rendirme y no dejar de luchar por lo que quiero. A Kamal, Javier y Alex por ser amigos y compañeros de muchas prácticas de la carrera y con los que he pasado mucho tiempo estos años. Echaré de menos esas partidas al futbolín después de comer con ellos. Y una mención muy especial a la personita que desde hace tiempo es dueña de mi corazón, te quiero mucho.

Un abrazo muy fuerte a todos.

RESUMEN

ROBONOVA-I es un robot humanoide que ofrece a profesores, estudiantes y amantes de la robótica un completo kit. Este robot puede caminar, correr, bailar, y una vez programado, competir en competiciones con otros robots.
Este proyecto describe la forma de conectar un sistema empotrado con los servos de ROBONOVA-I e implementar una librería que permita enviar instrucciones para interactuar con sus servos, como leer su posición o mover uno en una dirección específica.
La idea del proyecto también es dejar preparado un sistema para futuras mejoras como crear scripts de movimientos con restricciones de tiempo real, implementar algoritmos de inteligencia artificial o crear otras funciones para manejar los sensores del robot.

PALABRAS CLAVE

ROBONOVA-I, servo, sensor, sistema empotrado

ABSTRACT

ROBONOVA-I is a humanoid robot that offers educators, students and robotic hobbysits a complete robot package. This robot can walk, run, dance and once programmed, compete in robot competitions.
The project describes the way to connect a embedded system with the servos of the ROBONOVA-I and implemets a library that allow to send instructions to interact with the servos like read the position or move one in a specific direction.
Also the idea of the project is prepare a system to future improvements like make complex movement´s scripts with real time computing constraints, implement artificial intelligence algorithms or make other functions to manage the robot´s sensors.

KEY WORDS

ROBONOVA-I, servo, sensor, embedded system

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción y objetivos

1.1. Motivación

La motivación de este trabajo surje de la necesidad planteada por el Laboratorio de Robótica de la Universidad Carlos III de disponer de una interfaz sencilla para manejar los servos del robot Robonova I.

El trabajo consiste en la implementación de una librería en lenguaje C para la comunicación con servos del modelo Hitec HSR-8498HB. Con ello permitiremos crear subrutinas que controlen los servos sin necesidad de conocimientos altos de programación. Con esta librería se podrá ejecutar una batería de instrucciones que permita el movimiento de cualquier servo en la dirección que queramos.

Se parte también de la necesidad de crear una plataforma que utilice software libre como el caso de Linux, y por tanto se ha de instalar ese sistema operativo en la placa.

1.2. Objetivos

El objetivo del trabajo es implementar y diseñar una plataforma autónoma que controle varios servos utilizando como firmware una distribución de Linux.

Disponemos de una placa Ebox 3310MX-AP que hará las funciones de

control, envío y recepción de comandos. Por otra parte disponemos de un robot ROBONOVA-I compuesto de servos HSR-8498HB que se conectan a la placa.

La librería se programa en lenguaje C, para ser fácilmente reutilizable y ampliable en un lenguaje estándar. Se deja el código abierto para programadores que quiera añadir nuevas funcionalidades.

1.3. Estructura de la memoria

Una vez realizada la introducción, vamos a describir la estructura del presente trabajo:

- Capítulo 2 - Estado del arte: En el capítulo ponemos en contexto el trabajo realizado, explicando las tecnologías y componentes utilizados. Se compone de 6 secciones: introducción, robótica humanoide, servomotores, transmisión en serie, sistemas empotrados y Linux.
- Capítulo 3 - Análisis del sistema: En el capítulo se realiza el análisis de lo que debe realizar el sistema. El análisis consta de la toma de requisitos que especifican lo que hay que realizar en el trabajo.
- Capítulo 4 - Diseño. En el capítulo se detalla el diseño del sistema, desglosando el diseño del hardware, software y la librería de comunicación.
- Capítulo 5 - Implantación y pruebas. En el capítulo se cuenta la interconexión del sistema y se realizan las pruebas en base a los requisitos de software.
- Capítulo 6 - Planificación y presupuesto. En el capítulo se presenta la planificación llevada a cabo en el trabajo y el presupuesto elaborado.
- Capítulo 7 - Conclusiones y trabajos futuros. En el capítulo se exponen las conclusiones extraidas del trabajo, y las posibles mejoras y ampliaciones que se pueden realizar.
- Anexo A: Se incluyen los pasos y la documentación técnica para usar la librería.
- Anexo B: Se describen los pasos de la instalación de Lubuntu y Xenomai en la placa Ebox.

- Glosario: Se incluyen los términos poco conocidos acompañados de su correspondiente definición.
- Bibliografía: Se incluyen las referencias utilizadas para la realización del trabajo.

Capítulo 2

Estado del arte

En este capítulo describiremos los componentes y tecnologías en robótica humanoide y que vamos a utilizar en este trabajo. Las puntos que vamos a tratar son los siguientes:

- **Introducción**: esta sección pone en antecedentes al lector sobre aspectos de la robótica y su historia.
- **Robótica humanoide**: esta sección presenta los robots humanoides más empleados en docencia e investigación.
- **Servomotores**: esta sección cuenta las características técnicas de los servos y se presentan ejemplos de los más utilizados en robótica.
- **Transmisión en serie**: esta sección explica el funcionamiento de las transmisiones en serie.
- **Sistemas empotrados**: esta sección resume las características de los sistemas empotrados mostrando el hardware y sofware que utilizan.
- **Sistemas completos en un chip**: esta sección muestra las principales opciones comerciales en placas para robots.
- **Linux**: esta sección resume los aspectos principales del sistema operativo.

2.1. Introducción

Durante siglos, el hombre ha construido máquinas que imitaban partes del cuerpo humano. Los egipcios, construyendo brazos a las estatuas de sus dioses; los griegos, creando estatuas que operaban con sistemas hidráulicos, cuya utilidad era maravillar a los fieles que iban a rezar a los templos. En el siglo XVII en Europa se construyeron muñecos mecánicos muy ingeniosos que se asemejaban a los robots.

La robótica actual comienza en el siglo XVIII cuando en 1801 Joseph Jacquard inventa una máquina textil programable mediante tarjetas perforadas. La palabra robot fue introducida en la literatura en 1920, en la obra "Los Robots Universales de Rossum", escrita por el dramaturgo checo Karel Capek. Etimologicamente proviene de la palabra checa robota que significa labor forzada. Pero no es hasta 1939 cuando Isaac Asimov introduce el termino robotica, redactando sus tres famosas leyes:

1. *Un robot no puede actuar contra un ser humano o, mediante la inacción, que un ser humano sufra daños.*

2. *Un robot debe de obedecer las ordenes dadas por los seres humanos, salvo que estén en conflictoscon la primera ley.*

3. *Un robot debe proteger su propia existencia, a no ser que esté en conflicto con las dos primeras leyes.*

Pero no es hasta el desarrollo de los ordenadores a mediados del siglo XX cuando la robotica comienza un crecimiento exponencial. En 1961 el robot Unimate se instaló en la compañia Ford Motors para atender una máquina de fundición de troquel. Diez años mas tarde, en la universidad de Stanford se crea un pequeño brazo robótico de accionamiento eléctrico.

En la actualidad se desarrolan robots en muy diversos campos. Se desarrollan robots para ocio como el perro de Sony AIBO, uno de los juguetes mas sofisticados del mercado. En el entorno bélico podemos encontrar el robot CYPHER, un helicóptero robótico de uso militar. En el campo de la exploración espacial se emplean cada vez más. Uno de los más recientes es el famoso robot Curiosity que llego a Marte en agosto de 2012 para buscar indicios de vida en aquel planeta.

Los retos planteados por la robótica hoy en día son muy ambiciosos, y

pretenden dotar a los robots de algoritmos de inteligencia artificial para no convertirlos en meros autómatas, para expresar emociones, aprender...

2.2. Robotica humanoide

Nuestra atención se va a centrar en los robots mini-humanoides, cuyo principal uso está en el ámbito educativo y de investigación. Los dos ejemplares más populares son Robonova y Bioloid. Estos robots son capaces de realizar movimientos complejos y piruetas gracias a sensores que les alertan de los obstaculos. También se les puede controlar en remoto vía bluetooth o WIFI.

La Universidad los emplea para participar en CEABOT, un concurso de robots humanoides organizado por el Comité Español de Automática para alumnos de grado y postgrado de las universidades españolas. Las dos pruebas de este año fueron:

- **Prueba de movilidad**: El robot realizará una carrera de ida y vuelta salvando unos obstáculos colocados de forma aleatoria. En el segundo nivel, el robot realizará una carrera (solo ida), teniendo que superar una escalera con peldaños de 3 centímetros de altura.
- **Prueba de lucha**: Los combates consistirán en 3 asaltos de 2 minutos cada uno. Entre asalto y asalto habrá un tiempo máximo de 1 minuto. Ganará el combate el que haya ganado más asaltos. En caso de empate, se desempatará con la suma total de los puntos conseguidos. En caso de empatar a suma de puntos se hará un asalto más.

Estos robots se pueden adquirir en kits con todo lo necesario para su montaje y programación. Un kit dispone de los siguientes elementos:

- Un **procesador** con toda la electrónica de entrada y salida de los sensores resuelta. Además poseén un sistema operativo dentro del controlador (firmware), que eleva el nivel de programación de los procesadores, lo que posibilita el uso de lenguajes de alto nivel o interfaces gráficas para el desarrollo de la inteligencia de nuestro robot.
- Un conjunto de **sensores** que aprovechan la electrónica ya montada, como por ejemplo un sensor de luz donde el firmware interpreta el voltaje que entrega el sensor como un valor de 0 y 100.

- Un conjunto de **servos**, que pueden requerir o no alimentación complementaria, y que con el firmware podemos indicarle dirección, velocidad... sin necesidad de cálculos complejos.
- **Guia de construcción**, para poder montarlo facilmente y reutilizar componentes.

Las ventajas de usuar kits son:

- **Menor tiempo de construcción**: En pocas horas se puede tener un robots completamente montado para comenzar a utilizar.
- **Alta reusabilidad del material**: Podemos desarmar el robot y utilizar los servos, sensores, y demás piezas y con el procesador contruir un robot distinto.
- **Baja necesidad de conocimientos**: sin saber de electónica y poco de programación podemos desarrollar robots poderoros, aunque si se dispone de esos conocimientos, el aprovechamiento será mayor.

2.2.1. Robonova

Se trata de un robot desarrollado por la empresa coreana *Hitec Robotics*. El robots puede realizar movimientos como andar, bailar e incluso hacer volteretas.

Sus "músculos"son servos, desarrollados especialmente para este robot por la misma empresa. Poseé 16 servos, 5 en cada pierna y 3 en cada brazo con un ángulo de giro de 180º.

Tiene un microcontrolador MR-C3024 con CPU Atmel ATMEGA 128 de 8 bits y tecnlogía RISC. Poseé 40 puertos de entrada y salida digitales, puerto serie UART RS232 y 8 entradas analógicas. Con esta cantidad de de puertos se pueden controlar un gran número de dispositivos como sensores de distancia, giróscopos, displays LCD, sensores de infrarrojos, etc.

2.2.2. Bioloid

Bioloid es un kit robótico para docencia producido por la empresa coreana Robotis. Sus componentes lo forman unos pequeños servos llamados

Figura 2.1: El robot humanoide Robonova-I

Dynamixel AX-12A, los cuales se usan conectados en cadena (serie) para contruir robots con varias configuraciónes. Dispone de 18 servos, sensores de proximidad y luminosidad hacia delante y hacia los lados, un micrófono y un pequeño altavoz.

Contenido del kit:

- 1 x CM-5 (módulo controlador basado en el Atmel ATMega128 a 16 MHz)
- 18 x AX-12 (Servomotores Dynamixel controlados en serie)
- 1 x AX-S1 (módulo sensor del robot)
- Mas de 100 piezas mecánicas, ruedas, neumaticos para el ensamblaje con los servos (Comprehensive Frame Set)
- 1 x Puertos de conexión serie
- Batería recargable (9,6V, 2,3Ah, NiMH)
- Alimentador de potencia
- Cable serie RS-232 de 9 pins
- CD-ROM con Software de programación, vídeos, manuales, etc.
- Tornillería, tuercas, espaciadores

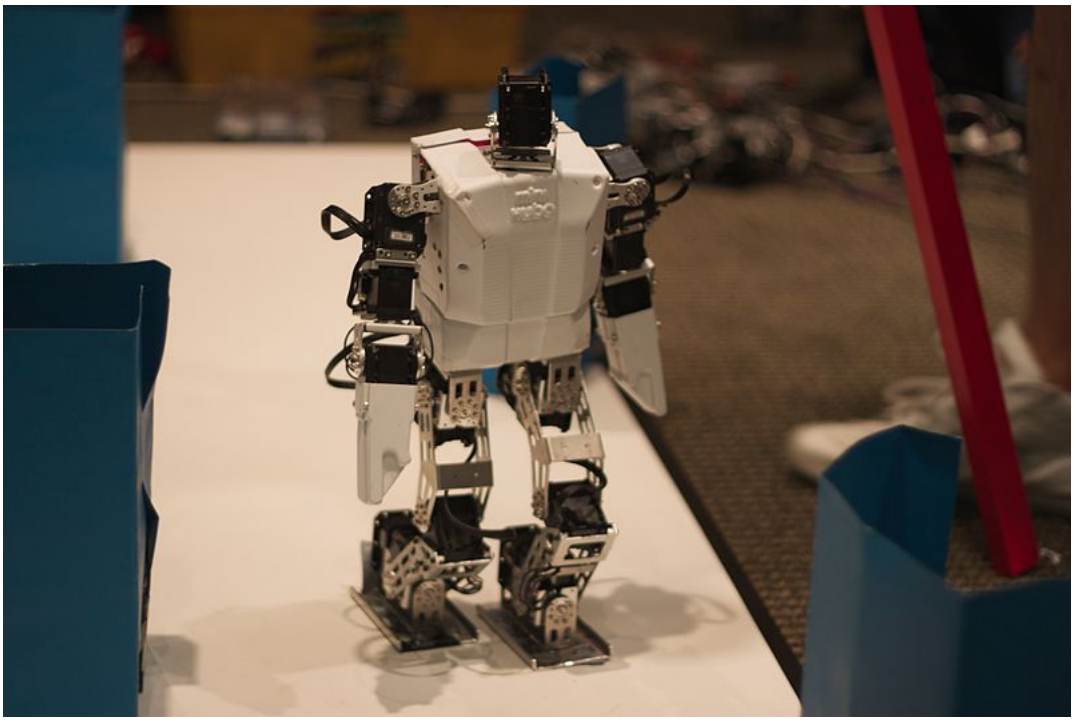

Figura 2.2: Robot humanoide Bioloid

2.3. Servomotores

Según la versión española de Wikipedia, un servomotor (también llamado servo) es un dispositivo similar a un motor de corriente continua que tiene la capacidad de ubicarse en cualquier posición dentro de su rango de operación, y mantenerse estable en dicha posición.

Los servos son muy útiles en robótica. Los motores son pequeños, y la circuitería de control es muy potente para su tamaño. Además no consumen mucha energía al ser proporcional la potencia a la carga que suele ser escasa.

A continuación se exponen las partes de un servo, se explica su forma de funionamiento, y por último se trata el tema del sumistro de energía.

Partes de un servo

Un servomotor está formado por:

- Carcasa: aloja todos los elementos.

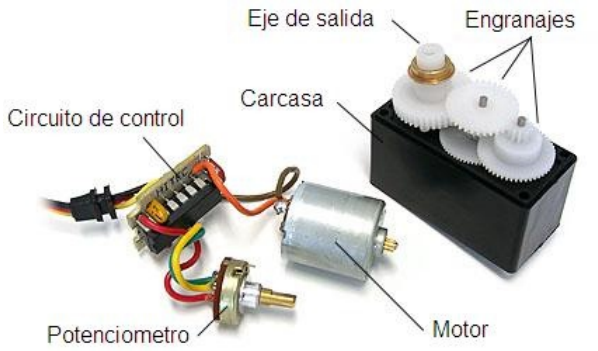

Figura 2.3: Un servo visto por dentro

- Motor: es un motor de corriente continua
- Engranajes: transfieren el par del motror al eje de salida, además reducen la velocidad del motor.
- Circuito de control: posiciona al eje y compara la señal que recibe del potenciometro interno con la señal de control externa.
- Potenciómetro: resistencia variable que se utiliza como sensor para conocer la posición del eje de salida.

Habitualmente, la capacidad de un servo es de 180 grados pero esté número dependerá del fabricante y modelo.

Los cables del servo son codificados en colores. Normalmente el cable blanco es el de la señal de control, el rojo el de voltaje (VCC) y el negro el de tierra (GND). Al disponer de un solo cable de control, transmisión y recepción no pueden hacerse de forma simultanea.

La descripción que hemos hecho se refiera a servos analógicos (que son los más comunes). Los servos digitales tienen un micro en su placa de control que analiza la señal, la procesa y controla el motor. Reaccionan mucho mas rápido y tienen mas fuerza pero son mas caros y consumen mas energía.

La señal de control del servo

El cable de control se usa para comunicar el ángulo. La señal de control es una onda cuadrada con una frecuencia generalmente de 50Hz y un valor de pico de entre 3 y 5V, conocida como PWM (Pulse Width Modulation). La frecuencia de control puede variar dependiendo del fabricante, pero esta

debe ser una señal estable para poder lograr un buen posicionamiento del eje de salida.

La posicón del eje del servomotor se controla variando el ancho del pulso de la señal de control. Si los circuitos dentro del servomotor reciben una señal de entre 0.5 a 1.4 milisegundos, éste se moverá en sentido horario; entre 1.6 a 2 milisegundos moverá el servomotor en sentido antihorario; 1.5 milisegundos representa un estado neutro para los servomotores estándares. A continuación se muestra un ejemplo de cada caso:

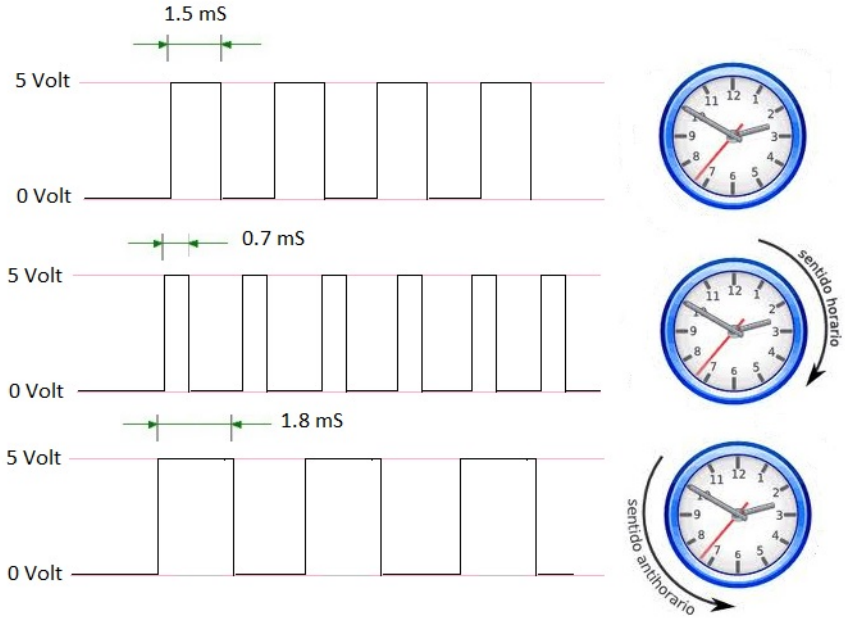

Figura 2.4: Duración de los impulsos y dirección obtenida

El periodo entre pulso y pulso (tiempo OFF) no es crítico, e incluso puede ser distinto entre uno y otro pulso. Se pueden emplear valores de entorno a 20 milisegundos (entre 10 y 30). Si el intervalo entre pulso y pulso es inferior al mínimo, puede interferir con la temporización interna del servo, causando un zumbido y vibraciones. Si es mayor que el máximo, el servo pasará a estado dormido entre pulsos. Esto provocará que se mueva con intervalos pequeños.

Para bloquear al servomotor en una posición determinada necesitamos enviarle todo el tiempo la señal con la posición que establezcamos. Así el sistema de control seguirá trabajando y el servo conservará la posición, resistiendose al movimiento que pueda venir de fuerzas externas y que intenten cambiarlo de posición.

Suministro de voltage

El voltage normal que proporcionan las compañias es un pack de 4 pilas de 1.2V de NiCd, que pueden dar hasta 4.8V. Esto puede variar en la práctica. Hay compañias que producen paquetes de pilas de 5 unidades, dando un voltaje de entre 6.5 - 7V cuando están completamente cargadas. La velocidad va a variar segun el voltaje que apliquemos. Consideramos 7V como un máximo seguro. Se puede usar también una fuente de voltaje de 5V.

La corriente que requiere depende del tamaño del servo. Normalmente el fabricante indica cual es la corriente que consume el servo. La corriente depende principalmente del torque usado y puede exceder más de un amperio si el servo está enclavado. Aunque lo mejor es siempre mirar las especificaciones propias de cada servo.

A continuación pasaremos a detallar las características técnicas de dos servos utilizados en robótica.

2.3.1. Hitec HSR-8498HR

HSR 8498HB es el servo motor digital de 54,7g empleado en el robot Robonova-I.

Características técnicas:

- Velocidad para 60º: 0,2 s a 6V / 0,18 s a 7,4V.
- Fuerza: 7,4 Kg/cm a 6V/ 9 Kg/cm a 7,4V.
- Dimensiones: 40 x 20 x 47 mm.
- Engranajes: Karbonite.

En la figura 2.5 podemos ver los componentes del servo.

2.3.2. Dynamixel AX-12+

Los servos Dynamixel AX-12+ son los servos empleados en el robot Bioloid. Poseen un microcontrolador que entiende 50 comandos, la mayoría para leer o fijar parametros que definan su comportamiento. Su peso es de 55g.

Figura 2.5: Servo motor Hitec HSR-8498HR

Características técnicas:

- Velocidad: 0.16sec/60º a 10V. (1024 velocidades configurables)
- Fuerza: 16.5kg.cm a 10V.
- Dimensiones:50 x 32 x 38mm.
- Engranajes: Engranajes y cuerpo de plástico.

2.4. Transmisión en serie

La transmisión en serie consiste en el envío de datos bits a bits, a diferencia de la transmisión en paralelo que transmite un mínimo de ocho bits de manera simultánea. Los bits se envían mediante señales eléctricas moduladas en amplitud. Los estándares mas conocidos para representar la información son el RS-232 y el TTL.

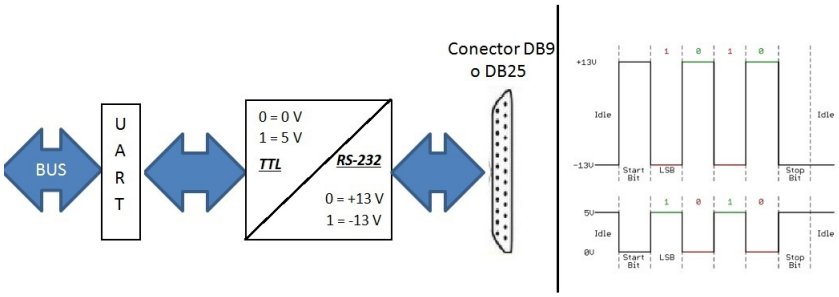

Figura 2.6: Tensión estándares RS-232 y diagrama de tiempo que muestra el envío de la señal 101

La imagen 2.6 muestra los niveles de tensión de ambos estándares y la representación del envío del paquete 101. Cómo vemos, para conectar entre si los dos puertos tenemos que tratar el problema de las distintos voltajes y la inversión de la señal. El RS-232 tiene un voltaje mayor que podría dañar los circuitos del microcontrolador serie. Hay varias soluciones para tratar estos problemas, el más común y facil es poner un adaptdor MAX-232 entre los dos dispositivos.

Los circuitos que implementan los estándares de transmisión en serie son los denominados UARTs, siglas de Universal Asynchronous Receiver-Transmitter.

La transmisión puede ser síncrona o asíncrona. La más utilizada es la asíncrona en la que los datos se envían carácter (byte) a carácter. La separación entre caracteres se consigue enviando un bit de comienzo, correspondiente al valor binario 0, y uno o varios bits de parada, correspondiente al valor binario 1.

La velocidad de trasmisión y recepción tiene que estar sincronizada. Esta velocidad se mide en bits por segundos o en baudios. En la emisión del bit de comienzo, emisor y receptor ponen en común sus relojes y mantienen la sincronía a lo largo de la transmisión del caracter.

Según el sentido de la transmisión, si esta se realiza de manera simultanea o no, se clasifica en tres tipos :

- **Simplex**: La información sólo discurre en un sentido y de forma permanente.
- **Half duplex**: La información discurre en ambos sentidos pero no de manera simultánea.
- **Full dúplex**: La información discurre en ambos sentidos y además puede hacerse de manera simultanea.

Para comprobar que los datos se transfieran sin interrupción, se permite el envió al final del caracter de un bit de paridad.

- **Paridad par**: El bit enviado tendrá un valor binario 0 si la suma de los valores binarios 1 en el caracter es par, y tendrá un valor binario 1 si la suma de los valores binarios 1 es impar.

- **Paridad impar**: El número de valores binarios 1 (datos más paridad) ha de ser impar.

Estandar RS-232C

El estandar RS-232 (Recommended Standard 232) fue introducido por primera vez en 1962 por la EIA (Electronic Industries Alliance). El RS-232C consituye la tercera revisión de la antigua norma RS-232, y cuyas diferencias entre ambas son mínimas. Existen dos tipos de conectores, el DB-25 de 25 pines, y la versión de 9 pines DB-9 que es el más difundido.

En la Figura 2.1 podemos ver el conector DB-9 cuyos pines de comunicación son los siguientes:

Pin	Nombre	Descripción
1	CD	Detección de portadora
2	RDX	Recepción de datos
3	TDX	Transmisión de datos
4	DTR	Terminal de datos preparado
5	GND	Tierra
6	DSR	Puesto de datos del sistema
7	RTS	Solicitud del envío
8	CTS	Borrado para enviar
9	RI	Indicador de llamada

Tabla 2.1: Pines DB-9

El conector DB9 se utiliza principalmente para conexiones en serie, ya que permite una transmisión asíncrona de datos según lo establecido en el estandar.

Universal Serial Bus

El Universal Serial Bus (USB) es un estandar industrial desarrollado en la decada de los noventa que define los protocolos, cables y conectores para conseguir la comunicación y alimentación electrica entre ordenadores y perfiféricos. La iniciativa del desarrollo partió de Intel y actualmente agrupa a mas de 685 compañias.

Figura 2.7: Logo USB 2.0

Existen cuatro tipos según su velocidad de transmisión:

- **1.0**: Baja velocidad, su tasa de transferencia es de hasta 188kB/s.
- **1.1**: Tasa de transferencia hasta 1.5MB/s.
- **2.0**: Tasa de transferencia de hasta 60 MB/s, pero con una tasa real 35MB/s. El cable USB 2.0 dispone de cuatro líneas, dos para datos y otras dos para alimentación.
- **3.0**: Tasa de transferencia de hasta 600MB/s. Se inclyen 5 contactos adicionales, y sigue siendo compatible con los estándares anteriores.

Los pines de comunicación del puerto USB 1.x/2.0 estandar son los siguientes:

Pin	Nombre	Color del cable	Descripción
1	VBUS	Rojo o naranja	+5V
2	D-	Blanco o dorado	Data -
3	D+	Verde	Data +
4	GND	Negro o azul	Tierra

Tabla 2.2: Pines USB 1.x/2.0

2.5. Sistemas empotrados

Un sistema empotrado es aquel que usa un ordenador para realizar una función específica, pero ni es usuado ni es percibido como tal. Se utilizan cada vez más en todo tipo de dispositivos: automoviles, televisores, electrodomésticos...

Las características de los sistemas empotrados son:

- Tienen una función concreta.
- Se integran dentro del dispositivo que controlan.
- Hardware limitado (CPU + RAM + memoria FLASH + elementos E/S).
- El software suele crearse a la vez que el dispositivo (Firmware).

Un ejemplo detallado se muestra en la figura 2.8. En esta figura se aprecia la CPU y memoria junto con una serie de interfaces que permite al sistema interactuar con el exterior. En el esquema se puede ver como existe una comunicación interior del sistema (a traves del bus de sistema) y una comunicación externa.

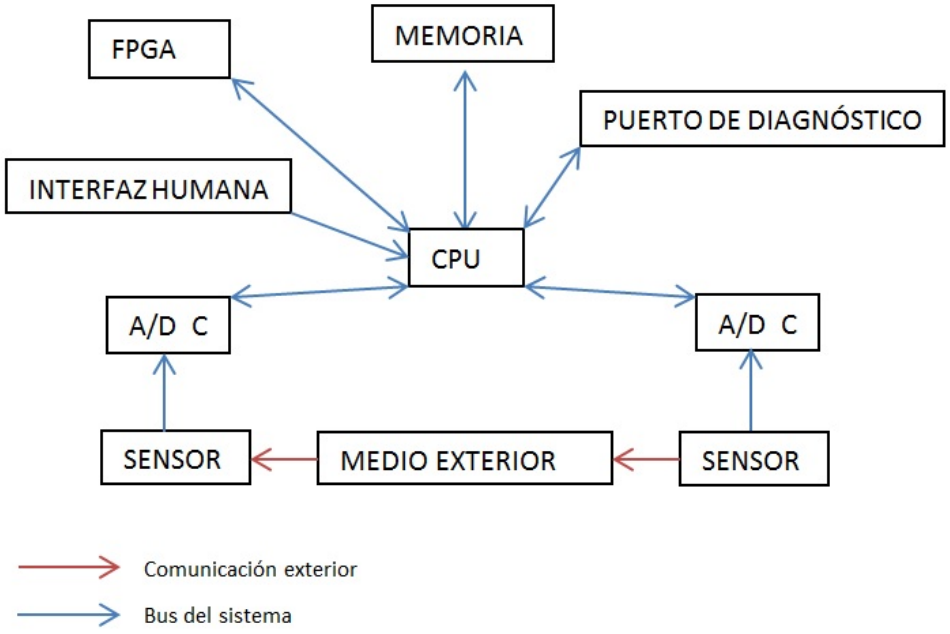

Figura 2.8: Arquitectura estandar de un sistema empotrado

2.5.1. Hardware para empotrados

Los tipo de arquitectura hardware son los siguientes, clasificados de menor a mayor potencia:

- Arrays de puertas programables (FPGAs).
- Microcontroladores.
- Sistemas completos en un chip.
- Placas madre.

FPGA

Una FPGA (del inglés Field Programmable Gate Array) es un dispositivo que contiene bloques electrónicos lógicos variados (puertas lógicas, biestables...). Su interconexión y funcionalidad pueden ser programadas. Permite crear sistemas completos en un solo chip y la tarea del programador es interconectar los diferentes elementos.

Microcontrolador

Un microcontrolador es un chip que incluye CPU, memoria y unidades de entrada y salida. Permiten un gran rango de posibilidades, permitiendo elegir un modelo ajustado al sistema que se desea reduciendo al máximo costes y consumo de energía. Un microcontrolador se programaba antiguamente en ensamblador pero actualmente se utilizan lenjuajes como ADA y C. El desarrolo se hace en otra máquina y se utilizan compiladores cruzados para generar el firmware.

El procesamiento más común en los robots pequeños es el provisto por microcontroladores. Los micros mas utilizados son los de la familia PIC, de la firma Microchip. La empresa Atmel fabrica todo tipo de microcontroladores, que tambien son de uso habitual en la construcción de robots. Su linea denominada AVR de 8 bits presenta características de bajo consumo, acrquitectura RISC y Harvard, 32 registros de 8 bits de proposito general y facilidad de implementación de lenguajes de alto nivel para la programación.

Sistemas completos en un chip

Los sitemas completos en un chip utilizan la misma filosofía que los microcontroladores, la única diferencia es la escala. Son mucho más potentes y suelen necesitar almacenamiento externo y dispositivos de entrada y salida externos más complejos.

Placas madre

Las placas madre son definiciones estandar de placas eléctricas para sistemas empotrados. Definen la forma de interconexión de elementos y las medidas, facilitando futuras ampliaciones consiguiendo abaratar costes. Ejemplos: Estandar PC104, micro ATX, mini ITX, etc.

2.5.2. Sistemas completos en un chip

Un sitema en un chip o SoC (del ingles system-on-chip) describe la tendencia cada vez mas frecuente de usar tecnologías de fabricación que integren todos o gran parte de los módulos componentes de un ordenador o cualquier sistema informático o electrónico, en un único circuito integrado o chip. Contrastan con los microcontroladores en la escala. Los microcontroladores tipicamente tienen menos de 100kB de RAM y frecuentemente son sistemas en un solo chip. Sin embargo, los SoC tienen procesadores más potentes capaces de ejecutar un sistema Windows ó Linix, los cuales necesitan de memoria externa (flash o RAM) para poder ser usados. Los SoC además pueden disponer de una gran variedad de periféricos. Algunos sitemas son demasiado complejos como para ser contruidos en un SoC, pasandose a utilizar los SiP (system in package) que componene de varios chips en un mismo paquete o placa. Otras opciones, como las que se utilizan en smartphones son las BeagleBoard, son los paquetes de paquetes que son apilados durante el montaje de la placa.

Estos sitemas constan de:

- Un microcontrolador, microprocesador o nucleo DSP. Algunos SoCs llamados sitemas en un chip multiprocesador (MPSoC) incluyen mas de un core.
- Módulos de memoria incluyendo la ROM, RAM, EEPROM y memoria flash.
- Generadores de frecuencia fija como por ejemplo osciladores y/o lazos de seguimiento de fase o PLLs.
- Componentes periféricos como contadores-temporizadores, temporizadores o relojes a tiempo real y generadores PoR(power-on reset, dispositivos que reajustan el sistema al recibir señal positiva, permitiendo a un sistema electrónico arrancar desde un estado conocido)
- Interfaces externas incluyendo estándares como USB, Ethernet, USART.

- Interfaces analógicas incluyendo ADCs(Conversor Analógico-Digital) y DACs (Conversor Digital-Analógico).
- Reguladores de voltaje y circuitos de Power Management.

Estos módulos estan conectados de acuerdo a estándares industriales de conexión de buses o también tecnologías propietarias como la arquitectura bus diseñada por ARM Ltd.

Entorno de ejecución

Un entorno de ejecución comprende el hardware y el software sobre el que se desarrolla la aplicación.

En los sistemas empotrados suele estar compuesto por:

- **Aplicacón**: que es el programa a ejecutar único.
- **Nucleo de ejecución**: que son el conjunto de librerias con la funcionalidad que requiere el programa y que está asociado con un lenguaje específico.
- **Sistema operativo**: se comunica con el hardware sobre el que se apoya el nucleo de ejecución

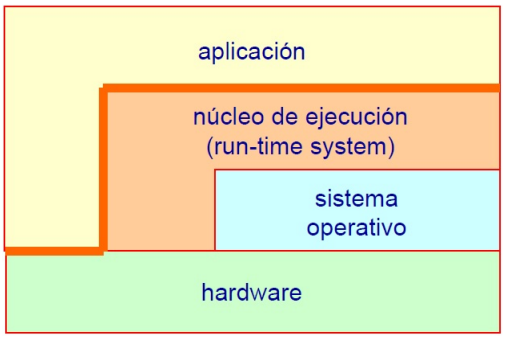

Figura 2.9: Entorno de ejecución de un Sistema Empotrado

Existen varias clases de entorno:

- **Entorno de ejecución en ADA**: Uno de los lenjuajes mas completos, dispone de un nucleo de ejecución propio y puede prescindir del sistema operativo.

- **Entorno de ejecución en JAVA**: Necesita una máquita virtual, teniendo un entorno de ejecución completo y por tanto pudiendo prescindir del sistema operativo.

- **Entorno de ejecución en C**: El lenguaje depende del sistema operativo que se use, y se reqiuere de la interfaz de este (por ejemplo POSIX), no teniendo nucleo de ejecución o siendo esté muy pequeño.

2.6. Linux

Es un sitema operativo basado en UNIX desarrollado bajo el modelo de sofware libre. El componente que define Linux es su kernel, cuyo primer programador fue Linux Torvalds que lo lanzó en 1991. El proyecto GNU, iniciado por Richard Stallman, creó las librerias, el compilador C y el principal sistema de soporte en el espacio de usuario, y por ello se prefiere que se refieran a él como GNU/Linux.

Parte de la interación entre el núcleo y los usuarios (o los programas de aplicación) se manejan habitualmente con las herramientas del proyecto GNU y con entornos de escritorios basados en , que también forma parte del proyecto GNU anuque tuvo un origen independiente.

De esta unión de programas y tecnologías surgen variantes, a las que se les adicionan diversos programas de apliciones de propósito específicos o generales y que se denominan distribuciones. Su objetivo consiste en ofrecer ediciones que cumplan con las necesidades de un determinado grupo de usuarios. Por ejemplo, para usuarios que quieran disponer de un sistema operativo que consuma pocos recursos se crean distribuciones como Lubuntu o Xubuntu. Otras son conocidas por su uso en servidores o supercomputadoras, donde tiene la cuota mas importante del mercado. Debido al bajo coste y a su facil personalización, Linux se usa frecuentemente en los sistemas empotrados.

Algunas de sus características destacadas son las mostradas en la tabla 2.3

Algunos de los componentes de un sistema Linux son:

- **Bootloader**: GRUB o LILO. Este programa es ejecutado en el ordenador cuando se enciende y carga el kernel de Linux a memoria.

Programado en	Varios, mayormente en C y en ensamblador
Familia	Unix
Modelo	Libre y de código abierto
Primer lanzamiento	1991
Última versión estable	3.9.7(20 Junio de 2013)
Última versión inestable	3.10-rc7 (22 Junio 2013)
Mercado	Pcs, moviles, sistemas empotrados, servidores...
Lenguajes	Multilenguaje
Tipo de kernel	Monolítico y multiusuario
Licencia	Muchas (Marca propoiedad de Linux Torvalds)

Tabla 2.3: Características de Linux

- **Programa init**: Este es un proceso lanzado por el kernel de Linux, y es la raiz del árbol de procesos. En otras palabras, todos los procesos son lanzados a través de él. Su proceso de comienzo es el servicio de login.
- **Librerias software**: Contienen código que puede ser usado por procesos en ejecución. Los sistemas Linux utilizan el formato ELF para los ficheros ejecutables, y un enlazador dinámico cuyo manejador de uso de las librerías dinámicas es "ld-linux.so". La librería de sofware mas común en los sistemas Linux es la librería GNU.
- **Programas de usuario**: Son los programas lanzados por la linea de comandos o en el entorno de ventanas.

Capítulo 3

Análisis del Sistema

En este capítulo se realiza un análisis detallado del sistema consistente en la toma de los requisitos de usuario, la representación de estos en casos de uso, y en su refinamiento para generar los de requisitos de software.

3.1. Requisitos de usuario

En esta sección se recogen los requisitos de usuario que se han considerado para la realización del trabajo. Estos requisitos constan de los siguientes atributos:

- **Identificador**: Descriptor único.
- **Nombre**: Título breve y descriptivo.
- **Descripción**: Comentario conciso y preciso sobre la funcionalidad requerida.
- **Necesidad**: Grado de necesidad de ese requisito. Se clasifica en esencial, deseable u opcional.
- **Prioridad**: Establece la prioridad para concretar el orden de realización del conjunto de requisitos. Se valora entre alta, media o baja.
- **Estabilidad**: A cada requisito se le asigna una probabilidad de cambio (alta, media o baja).

Los requisitos de usuario se dividen en dos grupos:

- **Requisitos de capacidad**: Especifican las funcionalidades deseadas a alcanzar.

- **Requisitos de restricción**: Establecen restricciones de cómo estos requisitos funcionales son implementados.

3.1.1. Requisitos de capacidad

Los requisitos de capacidad extraidos del ánalisis realizado son los siguientes:

RU.C.01	Medir corriente y tensión del servo.
Descripción	Se permitirá solicitar mediante la terminal de Linux la lectura de la corriente y de la tensión del servo mostrándose por pantalla.
Necesidad	Esencial
Prioridad	Alta
Estabilidad	Alta

Tabla 3.1: Requisito de capacidad 1

RU.C.02	Leer id y firmware del servo.
Descripción	Se permitirá solicitar mediante la terminal de Linux la lectura del identificador y firmware del servo mostrándose por pantalla.
Necesidad	Esencial
Prioridad	Alta
Estabilidad	Alta

Tabla 3.2: Requisito de capacidad 2

RU.C.03	Leer posición del servo.
Descripción	Se permitirá solicitar mediante la terminal de Linux la lectura de la posición mostrándose por pantalla.
Necesidad	Esencial
Prioridad	Alta
Estabilidad	Alta

Tabla 3.3: Requisito de capacidad 3

RU.C.04	Leer posición y establecer velocidad del servo.
Descripción	Se permitirá solicitar mediante la terminal de Linux la lectura de la posición y establecer una velocidad.
Necesidad	Esencial
Prioridad	Alta
Estabilidad	Alta

Tabla 3.4: Requisito de capacidad 4

RU.C.05	Mover el servo una posición determinada.
Descripción	Se permitirá solicitar mediante la terminal de Linux la posición a la que se desea que el servo gire.
Necesidad	Esencial
Prioridad	Alta
Estabilidad	Alta

Tabla 3.5: Requisito de capacidad 5

RU.C.06	Implementar conexión del servo con la placa.
Descripción	Se buscará un mecanismo para lograr conectar por el puerto serie la placa controladora que sustituya al microcontrolador del robot y el servo.
Necesidad	Esencial
Prioridad	Alta
Estabilidad	Alta

Tabla 3.6: Requisito de capacidad 6

3.1.2. Requisitos de restricción

Los requisitos de restricción son los siguientes:

RU.R.01	Utilizar Linux.
Descripción	El controlador llevará instalada una versión compatible de Linux.
Necesidad	Esencial
Prioridad	Alta
Estabilidad	Alta

Tabla 3.7: Requisito de restricción 1

RU.R.02	Controlar al menos dos servos simultáneamente.
Descripción	Se podrá mandar comandos a dos o más servos para que estos puedan hacer movimientos a la vez.
Necesidad	Deseable
Prioridad	Media
Estabilidad	Media

Tabla 3.8: Requisito de restricción 2

RU.R.03	Codificación de la librería en lenguaje C.
Descripción	La librería de comunicación se hará en el lenguaje C.
Necesidad	Esencial
Prioridad	Alta
Estabilidad	Alta

Tabla 3.9: Requisito de restricción 3

RU.R.04	Menu para controlar un servo.
Descripción	Realizar un programa en C que contenga un menú con las instrucciones implementadas para controlar un servo.
Necesidad	Opcional
Prioridad	Baja
Estabilidad	Baja

Tabla 3.10: Requisito de restricción 4

RU.R.05	Habilitar un entorno de desarrollo en tiempo real.
Descripción	Preparar el kernel de Linux instalado en el controlador para darle características de tiempo real.
Necesidad	Opcional
Prioridad	Baja
Estabilidad	Baja

Tabla 3.11: Requisito de restricción 5

3.2. Casos de Uso

En esta sección se muestran los casos de uso considerados en el Análisis del Sistema. Estos constan de los siguientes atributos:

- **Identificador**: Descriptor único.
- **Nombre**: Título breve y descriptivo.
- **Precondiciones**: Descripción del estado del sistema antes de la ejecución del caso de uso.
- **Escenario**: Secuencia de acciones que describe la funcionalidad.
- **Postcondiciones**: Descripción del estado del sistema después de la ejecución del caso de uso.

Los casos de uso extraidos son los siguientes:

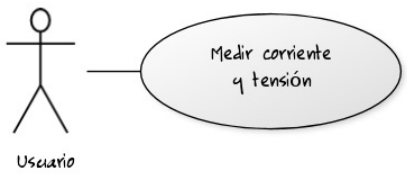

Figura 3.1: Casos de uso 1

CU.01	Medir corriente y tensión.
Precondiciones	Debe de estar conectado el servo al controlador.
Escenario	▪ Abrir una terminal ▪ Situarse en la ruta del ejecutable y escribir el siguiente comando: battery -p [puerto de conexión].
Postcondiciones	Por pantalla se muestra la tensión y el voltaje que devuelve el servo.

Tabla 3.12: Caso de uso 1

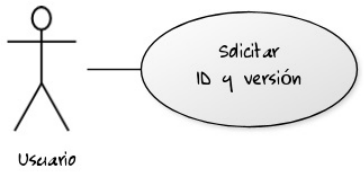

Figura 3.2: Casos de uso 2

CU.02	Solicitar ID y Versión.
Precondiciones	Debe de estar conectado el servo al controlador a través de un puerto habilitado.
Escenario	▪ Abrir una terminal. ▪ Posicionarse en la ruta del ejecutable y escribir el siguiente comando: version -p [puerto de conexión].
Postcondiciones	Por pantalla se muestra la versión y el ID del servo conectado.

Tabla 3.13: Caso de uso 2

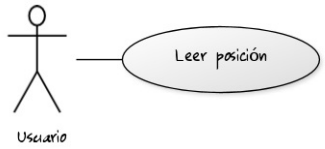

Figura 3.3: Casos de uso 3

CU.03	Leer posición
Precondiciones	Debe de estar conectado el servo al controlador a través de un puerto habilitado.
Escenario	▪ Abrir una terminal. ▪ Posicionarse en la ruta del ejecutable y escribir el siguiente comando: position -p [puerto de conexión].
Postcondiciones	Por pantalla se muestra la posición devuelta por el servo en grados.

Tabla 3.14: Caso de uso 3

Figura 3.4: Casos de uso 4

CU.04	Leer posición y establecer velocidad
Precondiciones	Debe de estar conectado el servo al controlador a través de un puerto habilitado.
Escenario	▪ Abrir una terminal. ▪ Posicionarse en la ruta del ejecutable y escribir el siguiente comando: speed -p [puerto] -i [identificador del servo] -s [velocidad].
Postcondiciones	Por pantalla se muestra la posición devuelta por el servo en grados.

Tabla 3.15: Caso de uso 4

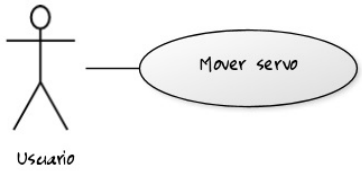

Figura 3.5: Casos de uso 5

CU.05	Mover un servo.
Precondiciones	Debe de estar conectado el servo al controlador a través de un puerto habilitado.
Escenario	▪ Abrir una terminal. ▪ Posicionarse en la ruta del ejecutable y escribir el siguiente comando: move -p [puerto] -i [identificador del servo] -a [posición en ángulos].
Postcondiciones	El servo se mueve a la posición indicada.

Tabla 3.16: Caso de uso 5

3.3. Requisitos Software

En esta sección se recogen los requisitos software que configuran el trabajo. Estos requisitos son el resultado del refinamiento de los requisitos de usuario y contan de los siguientes atributos:

- **Identificador**: Descriptor único.
- **Nombre**: Título breve y descriptivo.
- **Descripción**: Comentario conciso y preciso sobre la funcionalidad requerida.
- **Necesidad**: Grado de necesidad de ese requisito. Se clasifica en esencial, deseable u opcional.
- **Prioridad**: Establece la prioridad para establecer el orden de realización del conjunto de requisitos. Se valora entre alta, media o baja.
- **Estabilidad**: A cada requisito se le asigna una probabilidad de cambio (alta, media o baja).
- **Verificabilidad**: Establece la facilidad con la que se puede comprobar ese requisito (alta, media o baja).
- **Dependencias UR:**: Indica que requisitos de usuario se corresponden con el requisito sofware.

RS.01	Medir corriente y tensión del servo
Descripción	El sistema deberá proporcionar los valores de la tensión y corriente del servo conectado.
Necesidad	Esencial
Prioridad	Alta
Estabilidad	Alta
Verificabilidad	Alta
Dependencias UR	RU.C01

Tabla 3.17: Requisito de sofware 1

RS.02	Leer id y firmware del servo
Descripción	El sistema deberá proporcionar el identificador y firmware del servo conectado.
Necesidad	Esencial
Prioridad	Alta
Estabilidad	Alta
Verificabilidad	Alta
Dependencias UR	RU.C02

Tabla 3.18: Requisito de sofware 2

RS.03	Leer posición del servo
Descripción	El sistema deberá proporcionar la posición del servo conectado. Se le pasará el identificador del servo y se mostrará por pantalla la posición en ángulos en donde se encuntre el rotor.
Necesidad	Esencial
Prioridad	Alta
Estabilidad	Alta
Verificabilidad	Alta
Dependencias UR	RU.C03

Tabla 3.19: Requisito de sofware 3

RS.04	Leer posición y establecer velocidad del servo
Descripción	El sistema deberá proporcionar la posición del servo conectado y pasarle una velocidad al servo. Se le pasará el identificador del servo y la velocidad en un rango de 0 a 255. Se mostrará por pantalla la posición en ángulos en donde se encuntre el rotor.
Necesidad	Esencial
Prioridad	Alta
Estabilidad	Alta
Verificabilidad	Alta
Dependencias UR	RU.C04

Tabla 3.20: Requisito de sofware 4

RS.05	Mover el servo una posición determinada
Descripción	El sistema deberá permitir mover el rotor del servo a una determinada posición. Se introducirá el identificador del motor y el ángulo de posición, con un rango de -95 a 95 grados (números enteros) desde la posición en la que esté, no moviéndose en el caso de que se encuentre en la posición introducida.
Necesidad	Esencial
Prioridad	Alta
Estabilidad	Alta
Verificabilidad	Alta
Dependencias UR	RU.C05

Tabla 3.21: Requisito de sofware 5

RS.06	Implementar conexión para conectar el servo con la placa con Linux
Descripción	Se conectará el servo a la placa a través de un adaptador que convierta la señal de salida y entrada a señales TTL que entienden los servos. El sistema operativo empleado ha de ser Linux.
Necesidad	Esencial
Prioridad	Alta
Estabilidad	Alta
Verificabilidad	Alta
Dependencias UR	RU.C06 y RU.R.01

Tabla 3.22: Requisito de sofware 6

RS.07	Controlar al menos dos servos simultáneamente
Descripción	El sistema permitirá mover al menos dos servos simultaneamente, pudiendo variar tambien la velocidad de ambos.
Necesidad	Esencial
Prioridad	Alta
Estabilidad	Alta
Verificabilidad	Alta
Dependencias UR	RU.R.02

Tabla 3.23: Requisito de sofware 7

RS.08	Habilitar un entorno de desarrollo en tiempo real.
Descripción	Instalar framework parcheando el kernel de Linux, para que se permita crear aplicaciónes con restricciones de tiempo real.
Necesidad	Esencial
Prioridad	Alta
Estabilidad	Alta
Verificabilidad	Alta
Dependencias UR	RU.R.05

Tabla 3.24: Requisito de sofware 8

3.4. Matriz de trazabilidad

Expuestos los requisitos de sofware y de usuario se ha de comprobar que cada requisito de usuario se corresponda con al menos un requisito de sofware.

En la siguiente matriz se enfrentan ambos tipos de requisitos. Las correspondencias se marcan con una X. Si un requsito de usuario no tiene correspondencia con algún requisito de sofware, aparecerá un espacio en blanco en dicha casilla.

	RS.01	RS.02	RS.03	RS.04	RS.05	RS.06	RS.07	RS.08
RU.C.01	X							
RU.C.02		X						
RU.C.03			X					
RU.C.04				X				
RU.C.05					X			
RU.C.06						X		
RU.R.01						X		
RU.R.02							X	
RU.R.03								
RU.R.04								
RU.R.05								X

Figura 3.6: Matriz de trazabilidad

Capítulo 4

Diseño del Sistema

En este capítulo se especifica la arquitectura del sistema y el entorno tecnológico que le va a dar soporte, junto con la especificación detallada de los componentes a utilizar y la interacción entre ellos. Comenzaremos con un diseño general del sistema, desglosando despues el diseño hardware, sofware y de la librería. Este diseño servirá de guía en la fase de implementación.

4.1. Diseño General

Diagrama de Despliegue

Como se puede ver en la figura 4.1, el sistema consta de 3 nodos principales: el servo, la interfaz de conexión y la placa. El camino de comunicación de la información irá desde el controlador a cada uno de los servos, pasando por el adaptador que convertirá la señal de RS-232 a TTL. La respuesta de cada servo se hará en el sentido opuesto.

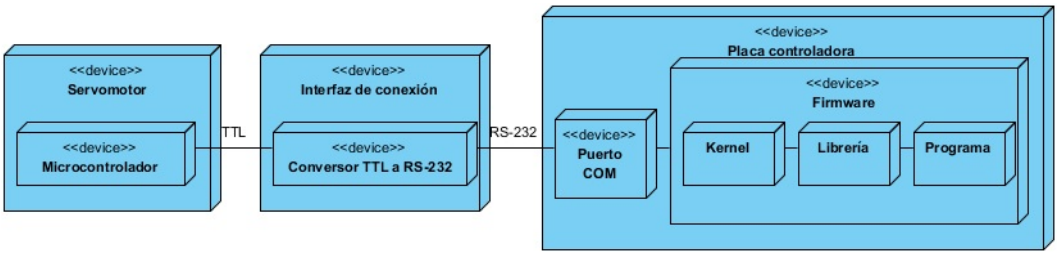

Figura 4.1: Diagrama de Despliegue del Sistema

A continuación explicamos con detalle la función de cada uno de los nodos:

- **Placa**: Contiene el firmware cuyo Kernel es el encargado de comunicarse con el hardware de la placa (Puertos, CPU, Memoria). Dentro de firmware está la librería de comunicación que hará de capa intermedia entre el programa y el kernel. La librería utiliza la API del estandar POSIX para hacer las llamadas al sistema propias de la lógica que vamos a implementar (llamadas a operaciones de lectura y escritura al puerto serie).
- **Conversor RS-232 a TTL**: Esta situado entre el controlador y el servo y es el encargado de convertir las señales electricas de tipo RS-232 del controlador a señales TTL que gestiona el servo.
- **Servo**: Es el componente que ejecutará las ordenes recibidas a través de su circuito de control, ya sea realizar un movimiento o enviar a la placa algún parámetro requerido (tensión, voltage, id, posoción...).

Interfaz de nivel de marco

Según la documentación aportada por Hitec, el servo intercambia datos a 19200 bits por segundos, con 8 bits por caratecter, 2 bits de parada y sin paridad.

El primer byte enviado por el contralador en la instrucción es la cabecera o byte de comienzo (0x080). Se muestra en la figura 4.2.

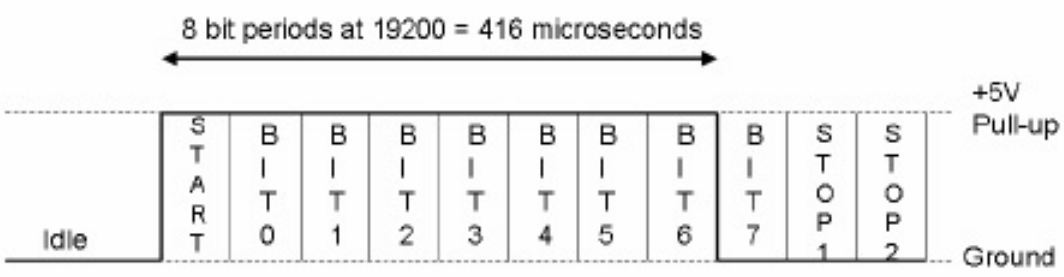

Figura 4.2: Esquema del paquete con los bytes de una instrucción

Si se tarda 1 segundo para enviar 19200 bits, para enviar un caracter de 8 bits se tardará:

8/19200 = 0.000416 segundos o 416 microsegundos.

Un paquete completo tiene 7 caraceres (bytes) y se compone de:

- Byte 1: Cabecera (0x080)
- Byte 2: Comando
- Byte 3: Dato enviado 1
- Byte 4: Dato recibido 2
- Byte 5: Checksum
- Byte 6: Dato recibido 1
- Byte 7: Dato recibido 2

El checksum se manda para asegurar la integridad del paquete y se calcula del siguiente modo: 256-(Byte 1 + Byte 2+ Byte 3 + Byte4)

Interfaz de comandos

Los comandos ofrecidos por Hictec son:

Leer ID y versión

- Byte 1: (0x080)
- Byte 2: (0xE7)
- Byte 3: (0x00)
- Byte 4: (0x00)
- Byte 5: 256-(Byte 1 + Byte 2+ Byte 3 + Byte4)
- Byte 6: Versión
- Byte 7: ID

Leer corriente y voltage

- Byte 1: (0x080)
- Byte 2: (0xE8)

- Byte 3: (0x00)
- Byte 4: (0x00)
- Byte 5: 256-(Byte 1 + Byte 2+ Byte 3 + Byte4)
- Byte 6: Corriente (0 a 255)
- Byte 7: Voltage (0 a 255)

Establecer velocidad del motor y leer posición

- Byte 1: (0x080)
- Byte 2: (0xE9)
- Byte 3: Servo ID (0x00 a 0x7F)
- Byte 4: Velocidad (0x00 a 0xFF)
- Byte 5: 256-(Byte 1 + Byte 2+ Byte 3 + Byte4)
- Byte 6: Posición Alta
- Byte 7: Posición Baja

La posición viene dada en pulsos de duración en milisegundos. El centro es 1500 y los extremos son 600 y 2400.

Establecer posición

- Byte 1: (0x080)
- Byte 2: (0x00 a 0x7F) = ID del servo
- Byte 3: Posición alta
- Byte 4: Posición baja
- Byte 5: 256-(Byte 1 + Byte 2+ Byte 3 + Byte4)
- Byte 6: (0x00)
- Byte 7: (0x00)

Diagrama de bloques

El diagrama de bloques de la figura 4.3 muestra el funcionamineto del código que llevará el programa que haga uso de la librería. Cuando el circuito esté conectado y el servo tenga la alimentación suficiente, se podrá comenzar a enviar las ordenes que están expuestas arriba. Los pasos para realizar dicho funcionamiento son los siguientes: Arrancar el controlador con el puerto COM activado. Configurar el puerto y enviar un paquete de instrucciones de 7 bytes al servo que deberá contestar enviando otro paquete del mismo tamaño con la información requerida, moviéndose o no dependiendo de la instrucción.

Figura 4.3: Diagrama de bloques del controlador

4.2. Diseño Hardware

En esta sección vamos a describir las diferentes alternativas de componentes hardware y el diseño de las conexiones.

Controlador

El controlador es el alma del sistema. Es el componente encargado de enviar la ordenen al servo y después recibir la respuesta de confirmación y

procesar esa información recibida, como por ejemplo su voltaje, identificador, posición...

Para la elección de este componente, debemos tener en cuenta ciertas consideraciones, entre las cuales destacan las siguientes:

- **Tamaño**: Es importante elegir un dispositivo con medidas reducidas para colocarlo en el robot y hacerlo mas portable.
- **Prestaciones**: La capacidad de cálculo y el número de puertos de entrada y salida de nuestro dispositivo es importante. Cuantas más funcionalidades tenga integradas menos componentes vamos a necesitar y por consiguiente estaremos cumpliendo el punto anterior y el siguiente.
- **Precio**: Como buscamos desarrollar un controlador de bajo coste, el precio de este dispositivo es importante en su elección.

De entre todos los fabricantes que proponen sus soluciones comerciales, nos hemos decantado por la placa Ebox 3310MX-AP desarrollada por DMP Electronic. Basado en el procesador Vortex, esta placa tiene en el chip todo el sistema (microprocesador, memoria, interfaces estandar...), conocido como SoC. Esta placa es capaz de ejecutar un sistema Linux o Windows y la utilizaremos para sustituir el controlador de la parte trasera del robot que viene de fábrica. Esta elección se debe a los siguientes motivos:

- Bajo consumo.
- Permite la instalación del Sitema Operativo Linux además de Windows.
- Fácil integración con el resto del circuito.
- Puertos estándar de comunicaciones (Dos puertos series COM).
- Reducido tamaño.
- Bajo costo de implementación.

Estos motivos hacen de este dispositivo empotrado una buena elección para su uso como controlador.

En la figura 4.4 podemos ver una foto de este dispositivo. Con la placa

Figura 4.4: Ebox 3310MX-AP

realizaremos la instalación del sistema operativo, la programación de la librería y realizaremos las pruebas necesarias para comprobar la comunicación con los servos.

Servo

Se han propuesto la posibilidad de utilizar dos servos para el trabajo, Dinamixel AX-12A y Hitec HSR-8498HR. Se ha optado por trabajar con los servos Hitec debido a 3 razones:

- Menor coste: (43.65 Euros frente a 52.90 de Dinamixel)
- Mayor facilidad de implementación: Dispone de un menor número de instrucciónes y estan mejor detalladas en la documentación.
- Conexión en paralelo: Cada servo se conecta independientemente, pudiendo hacer pruebas independientemente sin la necesidad de conectar uno a otro. En Dinamixel, la intrucción va pasando por cada servo (conexión serie) hasta llegar al servo de destino de la instrucción.

Una vez elegido, pasaremos a detallar los cables de conexión del servo. Cada servo dispone de tres cables, como se muestra en la figura siguiente:

Figura 4.5: cmd: Cables de un servo.

El cable de arriba (GND) es el que va conectado a tierra. El central es el de alimentación (VCC) y va conectado a la entrada de 6V. El cable gris (el que aparece abajo del todo) es el de datos y es el encargado de transmitir y recibir los paquetes.

Conversor RS-232 a TTL

Se barajaron como alternativas de conexión la utilización de los puertos COM y USB de la placa.

Se ha optado por utilizar el puerto COM para dejar disponible el USB por si en un futuro quisieramos conectar más de un hardware (tarjeta WIFI, memoria externa, conector bluetooth, ratón etc...).

Para conectar el servo al PC se necesita un interfaz que transforme la señal del puerto RS-232 a señales TTL. Los conversores estándar RS-232 a TTL invierten la señal y eso no es correcto para el protocolo HMI que utiliza el servo. Por este motivo, se opto por confeccionar el adaptador a medida, soldando los pines necesarios, y utilizando una resistencia que regulase el voltage de salida para adaptarlo al del servo.

La conexión eléctrica al servo se realiza de la siguiente manera:

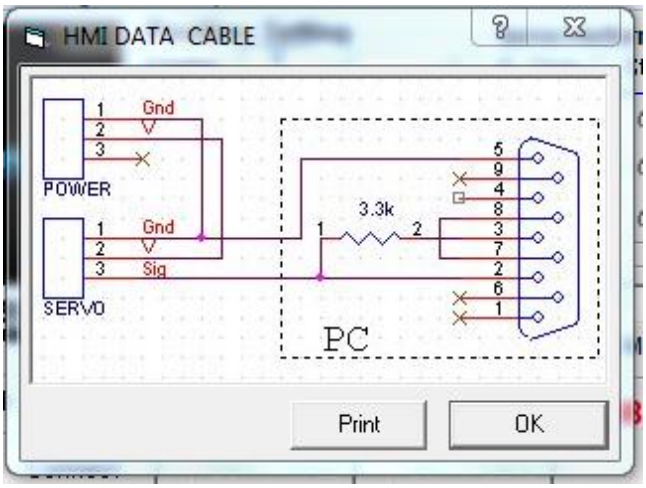

Figura 4.6: cmd: HMI data cable.

Se compró un conector DB9 hembra, una resistencia de 3.3K Ohm, y 4 cables cortos. Los cables se soldaron a los pines del conector como aparecen en la figura 4.6. Para hacer funcionar los servos se necesita alimentación de 6 a 7.7 Voltios. Para esto, se consiguió un transformador de 6 Voltios que suministrase ese voltaje mínimo. El protocolo HMI permite conectar múltiples servos al mismo bus a la vez, cableando todos los pines juntos. Para hacer las conexiones de forma cómoda se dispone de una protoboard, quedando la conexión de ejemplo de un servo como en la figura 4.7

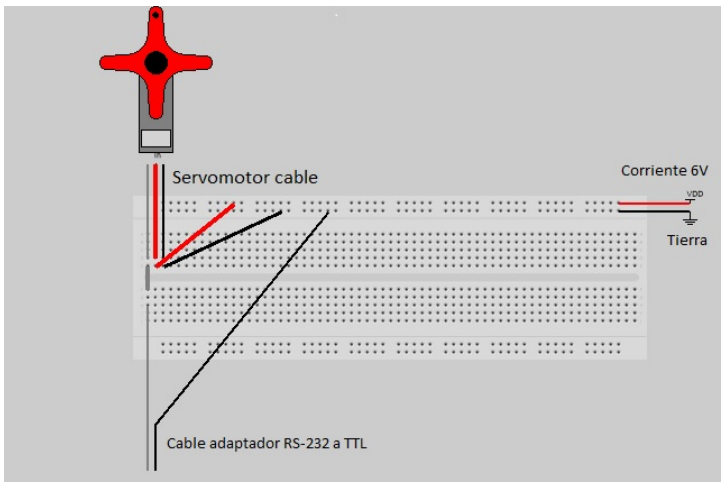

Figura 4.7: cmd: Conexiones del circuito montado en una protoboard.

4.3. Diseño Software

En esta sección se detalla el diseño sofware que dará apollo a la librería y que permitirá el control del servo. Esto se traduce en la selección del entorno de ejecución y del sistema operativo. Al final de la sección se explica la configuración del puerto serie con la interfaz POSIX y la conversión de ángulos a milisegundos.

4.3.1. Selección de la distribución de Linux y del entorno de ejecución

Uno de los requisitos del sistema es utilizar software libre en el controlador. Para seleccionar la versión de Linux se tuvieron en cuenta las características técnicas de la placa. Al tener caracteristicas modestas, se opto por buscar una versión de Linux que no consumiera mucha memoria RAM. Se contemplaron las versiones de Lubuntu y Ubuntu y se elegió la primera al requerir poca memoria. Además su instalación es sencilla y dispone de un amplio catalogo de sofware y de documentación para instalar en la placa.

Según la documentación técnica de Lubuntu, con 512MB de RAM se obtiene un buen rendimiento. Recordemos que el procesador es un Vortex86MX+ a 1Ghz con arquitectura i586, modesto pero suficiente para ejecutar fluidamente la distribución de Lubuntu.

La versión de Lubuntu elegida es la 10.04 LTS debido a que es la última versión que reconoce la arquitectura i586 implementada en la placa. Versiones posteriores son optimizadas para la aquitectura i686 que hace uso de instrucciones CMOV que no estan presentes en el prosador Vortex86MX y por tanto no funcionan.

Con estas caracteristicas, la placa Ebox es capaz de ejecutar un sistema operativo de tiempo real. Es posible por tanto instalar un framekwork dedicado a ello como Xenomai, que coopera con el kernel de Linux para proveer de una interzaz de tiempo real a las aplicaciones en el espacio de usuario.

Con respecto al entorno de ejecución, se ha elegido el de C por varios motivos:

- Es un lenguaje muy utilizado y que hemos manejado a lo largo de la carrera.

- Compatible con Linux y con la interfaz POSIX que nos facilita la comunicación con el puerto serie.

4.3.2. Configurar el puerto Serie con POSIX

Desde que el puerto serie trabaja como si fuera un fichero, la función *open* se usa para acceder a él.

```
int fd; /* Descriptor de fichero del puerto */
fd = open("/dev/ttyS0", O_RDWR | O_NOCTTY | O_NDELAY);
```

Para leer y escribir en él se usan las llamads al sistema propias *write* y *read*. Cuando dejamos de utilizar el puerto, se cierra con la función *close*.

Para cambiar los parametros del puerto serie(baudrate, parity...), Linux soporta la interfaz POSIX. Lo primero que necesitamos es incluir el fichero de cabeceras ¡termios.h¿. Este fichero define la estructura de control de la interfaz de entrada y salida del terminal, ademas de las funciones de control de POSIX.

Las dos funciones más importantes en POSIX son *tcgetattr* y *tcgetattr*. Funcionan como getter y setter respectivamente para configurar la estructura con las opciones del puerto que establezcamos.

En la tabla 4.1 se muestran los atributos de la estructura.

Atributo	Descripción
c_lflag	Opciones lineales
c_iflag	Opciones de entrada
c_oflag	Opciones de salidas
c_cc	Caracteres de control
c_ispeed	Baudios de entrada(nueva interfaz)
c_ospeed	Baudios de salida(nueva interfaz)

Tabla 4.1: Atributos de la estructura Termios

A través de operadores a nivel de bits (negaciones, AND y OR) se consiguen establecer los valores de estos atributos con la configuración requerida.

Opciones de control

El atributo *c_cflag* controla la tasa de baudios, numero de bits de datos, paridad, bits de parada y flujo de control del hardware. Las configuraciónes establecidas según los requerimientos del servo son:

Establecer la tasa de baudios (Según la documentación técnica de Hitec a 19200 bits/segundo)

Las funciones *cfsetospee* y *cfsetispeed* son las encargadas de establecer la tasa de baudios en la estructura *termios*. En código se traduce en:

```
struct termios options;

/*
 * Obtener las opciones del puerto
 */
tcgetattr(fd, &options);
/*
 * Establecer el baud rates a 19200
 */
cfsetispeed(&options, B19200);
cfsetospeed(&options, B19200);
```

Establecer tamaño de carácter

No hay funciones para establecer el tamaño y tenemos que establecerlo en un atributo. Para establecer 8 bits de tamaño, en el código hacemos lo siguiente:

```
options.c_cflag &= ~CSIZE; /* Máscara */
options.c_cflag |= CS8;    /* Establece 8 bits de datos */
```

Establecer chequeo de paridad

Como el tamaño de caracter, hay que especificar manualmente la paridad y el tipo. Para nuesro servo, sin paridad con 2 bits de parada. En el código queda así:

```
options.c_cflag &= ~PARENB /* sin paridad*/
options.c_cflag &= CSTOPB /* 2 bits de parada*/
```

4.3.3. Conversión de ángulos en segundos

Para convertir los ángulos del servo (AS) en ancho del pulso (AP) utilizamos la siguiente formula:

AS = (AP-1500) / 10

Despejando obtenemos que los pulsos son:

AP = (AS*10) + 1500

Por ejemplo, para un ángulo de 90º, el ancho del pulso será:

AP = (90 * 10) + 1500 = 2400ms.

La posición dada por el ancho del pulso es almacenada en un dos bytes, y por tanto los milisegundos obtenidos los tenemos que dividir en dos partes. Para el ejemplo anterior, los 2400 milisegundos se traducen en:

posición alta = 24 = 18(en hexadecimal)

posición baja = 00 = 0 (en hexadecimal)

Para programarlo, se utilizan operadores de desplazamiento de bits y máscaras para quedarnos con los bits apropiados. En lenguaje c se traduce en:

char posicionAlta = (unsigned char)((posicion8) & 0xFF);

char posicionBaja = (unsigned char)posicion & 0xFF;

Para la parte alta, se desplazan los bits ocho posiciones a la derecha, y después se realiza una suma lógica con una máscara de un byte que tiene todos los bits a uno, guardándose en un char. Así conseguimos quedarnos con la parte alta y almacenarla en un byte.

Para la parte baja, simplemente se hace una suma lógica de los bits haciendo una conversión a char para almacenarlo.

4.4. Diseño de la Librería

En esta sección vamos a plantear la interfaz entre el usuario y los servos del robot. Esto significa, una conexión funcional que el usuario pueda utilizar para realizar moviemientos a través de las instrucciones de control que se implementen en la librería.

Elección de atributos y métodos

Según los requisitos de capacidad extraidos de la fase de análisis, y conociendo las instrucciones con las que trabajan los servos, podemos realizar un análisis de las clases y métodos que llevará nuestra librería de comunicación. Se ha optado por crear una sola clase que con los siguiente variables y métodos:

Variables globales

Variable	**Valor**	**Descripción**
BAUDRATE	B19200	Tasa de bits
HSR_BUFFER_IN_SIZE	7	Buffer de lectura
HSR_BUFFER_OUT_SIZE	7	Buffer de escritura
B_VERSION	0xE7	Versión e ID
B_BATTERY	0xE8	Corriente y voltaje
B_Read_ADC_pos	0xE5	Lectura de la posicion
B_ID_R_POS_PC	0xE9	Selección de velocidad
B_ID_W_MOV_MAX	0	Movimiento de un servo
B_Write_CMD	0xE6	Movimiento a todos los servos

Variables locales

Variable	**Descripción**
unsigned char data_in[HSR_BUFFER_IN_SIZE]	Array de lectura
unsigned char data_out[HSR_BUFFER_OUT_SIZE]	Array de escritura
int fport	Descriptor del puerto

La variable BAUDRATE contiene el valor necesario de velocidad para la comunicación por el puerto serie. La variable fport estará incializada a 0. Cada instrucción guarda el valor en hexadecimal que entiende el microcontrolador del servo para realizar la funcionalidad requerida.

Métodos

- **int CommResource(char *argv)** Funcionalidad: Abre el puerto COM y establece los atributos necesarios para enviar y recibir paquetes.

  Valores de entrada: Puerto COM pasado por parámetro desde la terminal.

  Valor devuelto: Descriptor del puerto abierto para lectura/escritura.

- **sendPacket** Funcionalidad: Envía por el puerto COM un paquete del tamaño que se pase como parámetro.

  Valores de entrada: Descriptor del puerto, paquete, tamaño.

  Valor devuelto: Tamaño del paquete enviado .

- **receivingPacket** Funcionalidad: Recibe por el puerto COM un paquete del tamaño que se pase como parámetro.

  Valores de entrada: Descriptor del puerto, paquete, tamaño.

  Valor devuelto: Tamaño del paquete recibido.

- **getIDVersion** Funcionalidad: Llena el array de escritura con los bytes correspondiente de la instrucción y llama al método sendPacket para enviarlo y a continuación a receivingPacket para recibir el paquete de respuesta, llenando el array de lectura e imprimiendo su contenido.

  Valores de entrada: Ninguno.

  Valor devuelto: Ninguno.

- **getCurrentVoltage** Funcionalidad: Llena el array de escritura con los bytes correspondiente para obtener la corriente y voltage y llama al método sendPacket para enviarlo y a continuación a receivingPacket, llenando el array de lectura e imprimiendo su contenido.

  Valores de entrada: Número de argumentos.

  Valor devuelto: Ninguno.

- **getPosition** Funcionalidad: Llena el array de escritura con los bytes correspondiente para obtener la posición y llama al método sendPacket para enviarlo y a continuación a receivingPacket, llenando el array de lectura e imprimiendo su contenido.

  Valores de entrada: Número de argumentos.

  Valor devuelto: Ninguno.

- **setMotorSpeedReadPosition** Funcionalidad: Llena el array de escritura con los bytes correspondiente para establecer velocidad y leer posición y llama al método sendPacket para enviarlo y a continuación a receivingPacket, llenando el array de lectura e imprimiendo su contenido.

  Valores de entrada: Número de argumentos, id y velocidad.

  Valor devuelto: Ninguno.

- **setPosition** Funcionalidad: Llena el paquete con los bytes correspondiente para establecer la posición del servo y llama al método sendPacket para enviarlo y a continuación a receivingPacket imprimiendo su contenido.

  Valores de entrada: Número de argumentos, id y posición.

  Valor devuelto: Ninguno.

- **setAllPosition** Funcionalidad: Llena el array con los bytes correspondiente para establecer la posición de todos los servos y llama al método sendPacket para enviarlo y a continuación a receivingPacket, llenando el array e imprimiendo su contenido.

  Valores de entrada: Número de argumentos, id y posición.

  Valor devuelto: Ninguno.

- **RxTxchk** Funcionalidad: Recibe los 4 bytes de la instrucción indicados para calcular su checksum.

  Valores de entrada: 4 bytes.

  Valor devuelto: byte de checksum.

Capítulo 5

Implantación y pruebas

En este capítulo vamos a presentar la integración final del sistema y los resultados de las pruebas realizadas.

5.1. Implantación

Para seguir los pasos de la instalación del firmware (sistema operativo), de Xenomai y para habilitar el puerto COM desde la BIOS, leer el Apéndice B.

Para llevar a cabo la implantación se necesita el robot Robosavi-I, una protoboard, la placa Ebox, una fuente de alimentación de 6V y el conversor RS-232 a TTL fabricado. Para especificar los comandos que queremos realizar se requiere de un monitor, ratón y teclado. La conexión se realiza conectando la placa al monitor con un cable VGA, el ratón por USB y teclado por USB o PS2. A continuación se introduce la tarjete SD en la placa con Lubuntu instalado y configurado, y se enciende para arrancar el sistema. Después conectaremos el conversor al puerto COM1, con los pines de tierra y datos conectados a la protoboard. Los pines del servo o servos a controlar los conectamos a la protoboard, teniendo cuidado en conectar el cable de alimentación, datos y tierra en su correspondiente lugar. Para finalizar, abrimos una terminal y vamos a la ruta donde este el fichero binario, en mi caso: en la carpeta /home/roboard/workspace. Crearemos por cada comando un enlaces simbólico al ejecutable y lo ejecutaremos con los parametros requeridos (ej: version -p ttyS0).

5.2. Problemas

En la parte hardware, el principal problema encontrado fue comunicar el servo con la placa. El adaptador que se compró al principio para convertir las señales rs232 a ttl no valía porque tenía un circuito MAX232 que invertía la señal, y por eso cuando se enviaba un paquete al servo este no enviaba nada al controlador. Se solucionó confeccionando uno propio según las indicaciones del datasheet del servo. Otro de los problemas fue sincronizar la velocidad de controlador y servo. El servo trabaja a 19200 baudios y la velocidad del puerto por defecto en la placa era de 115200. Por eso, se tuvo que acceder a la BIOS para cambiar la velocidad. En la instalación de Linux en la SD se tuvo cuidado en escoger una tarjeta de calidad ya que leyendo la documentación de la placa se encontró que una tarjeta de mala calidad o lenta podía probocar problemsa de rendimiento o corrupción de datos (que el sistema dejara de arrancar). En este caso se optó por comprar una de Clase 10 de un fabricante de prestigio como Sandisk para ahorrarnos problemas en el futuro.

En la parte sofware, la dificultad mayor estuvo en conseguir que Linux reconociera el driver que leyera la tarjeta SD. El método para solucionar esto fue realizar la instalación de Linux en una máquina virtual con la tarjeta SD conectada al PC. Después se bajaron las fuentes de kernel apropiadas y se habilitó las opciones del driver antes de compilarlo. Una vez echo arrancaba el sistema desde la SD en la placa EBOX sin problemas. Todos estos pasos pueden verse en el Apéndice B.

5.3. Pruebas

Se realizan dos tipos de pruebas, de integración y de aceptación. Las pruebas de integración comprueban el comportamiento del sistema una vez conectado y las pruebas de aceptación verifican que se cumplan todos los requisitos extraidos en la fase de análisis. Las pruebas constan de los siguientes atributos:

- **Identificador**: Descriptor único.
- **Nombre**: Título breve de la prueba a realizar.
- **Descripción**: Comentario detallado de la prueba.

- **Procedimiento**: Pasos a seguir para realizar la prueba.
- **Requisito software relaccionado**: Muestra a que requisito se corresponde (sólo para pruebas de aceptación)
- **Resultado**: Muestra el resultado obtenido al realizar la prueba.
- **Estado**: Muestra si la prueba se ha realizado con éxito.

5.4. Pruebas de integración

Las pruebas de integración realizadas son las siguientes:

PRU.I.01	Medir tensión del circuito
Descripción	Se mide la tensión existente en el servo cuando está conectado al microcontrolador
Procedimiento	Utilizar un multimetro, colocándolo en la posición de corriente continua en la escala de 20V. Se coloca la punta de tierra de color negro sobre el pin de tierra del servo y la punta de tensión sobre el pin de tensión(cable central).
Resultado	La tensión del circuito es de +6.2V
Estado	Prueba realizada con éxito.

Tabla 5.1: Prueba de integración 1

PRU.I.02	Comprobar conexión del servo al puerto COM
Descripción	Comprobar que se tiene habilitado en la BIOS el puerto COM con el baurate necesario y ver si está activo.
Procedimiento	Ir a la BIOS al iniciar la placa pulsando F10. En el menú chipset, verificar que el baudrate del puerto COM1 está a 19200 y sino cambiarlo con el cursor y dando a intro. Salir de la BIOS guardando los cambios con F12. Para verificar que se tiene activo el puerto COM1 escribir en una terminal: setserial g /dev/ttyS*. Deberá aparecer el puerto COM que hemos configurado en la BIOS.
Resultado	Se muestra por pantalla lo siguiente: /dev/ttyS0, UART: 16550A, Port: 0x03f8, IRQ: 4
Estado	Prueba realizada con éxito.

Tabla 5.2: Prueba de integración 2

5.5. Pruebas de aceptación

Las pruebas de aceptación las realizaremos con el servo 1 del robot conectado. Las pruebas hechas son las siguientes:

PRU.A.01	Leer corriente y tensión de un servo
Descripción	Se solicita la corriente y tensión que le llega al servo 1
Procedimiento	▪ El usuario ejecuta el comando: ./battery ttyS0
Resultado	Obtenemos por la terminal la corriente y tensión del servo en hexadecimal: Current: 0x1D Voltage: 0x54
Requisito capacidad	RS.01
Estado	Prueba realizada con éxito.

Tabla 5.3: Prueba de aceptación 1

PRU.A.02	Leer id y versión
Descripción	Se solicita el identificador y versión del servo 1
Procedimiento	El usuario ejecuta el comando: ./version ttyS0
Resultado	Obtenemos por la terminal el id y la versión: Versión: 0x35 ID: 0x01
Requisito capacidad	RU.02
Estado	Prueba realizada con éxito

Tabla 5.4: Prueba de aceptación 2

PRU.A.03	Leer posición
Descripción	Se solicita la posición del servo 1
Procedimiento	Se coloca el rotor del servo en la posición 0, que esta indicado en el rotor. El usuario ejecuta el comando: ./position ttyS0
Resultado	Obtenemos por la posición: Position degree: 0
Requisito capacidad	RU.03
Estado	Prueba realizada con éxito

Tabla 5.5: Prueba de aceptación 3

PRU.A.04	Leer posición y establecer velocidad del servo
Descripción	Se solicita la posición de nuevo y se establece una velocidad de rotación. En este caso probaremos con la mínima que es 1.
Procedimiento	Con el rotor en la posición 0, el usuario ejecuta el comando: ./speed ttyS0 1 1
Resultado	Obtenemos por la posición: Position degree: 0.
Requisito capacidad	RU.04
Estado	Prueba realizada con éxito

Tabla 5.6: Prueba de aceptación 4

PRU.A.05	Mover el servo una posición determinada
Descripción	Se solicita al servo moverse de la posición 0 a la +90
Procedimiento	Se coloca una cinta con un celo sobre la posición 0 del servo que está dibujada sobre el rotor. El usuario ejecuta el comando: ./move ttyS0 1 90
Resultado	Comprobamos como la cinta del servo se mueve lentamente hacía la izquierda parándose cuando se realiza el movimiento de 90 grados.
Requisito capacidad	RS.05
Estado	Prueba realizada con éxito

Tabla 5.7: Prueba de aceptación 5

PRU.A.06	Implementar conexión para conectar el servo con la placa Ebox
Descripción	Conectar el circuito
Procedimiento	Se conecta la placa al servo con el conversor RS-232 conectado al puerto COM1 de la placa.
Resultado	La conexión se hace correctamente
Requisito capacidad	RS.06
Estado	Prueba realizada con éxito

Tabla 5.8: Prueba de aceptación 6

PRU.A.07	Mover dos servos a velocidades distintas.
Descripción	Poner los brazos del robot en posición horizontal. Mover los servos numero 7 y 11 del brazo en 45 grados hasta llegar al angúlo 90 y volver a la posición inicial del mismo modo. El primer movimiento se hará a la velocidad mas alta, el segundo a la media y el tercero a la más baja.
Procedimiento	Se crea y se ejecuta un shell script con el nombre script_movimientos_lab. Para ejecutarlo, hay que ponerse en la ruta donde se encuentre el ejecutable y escribir ./script_movimientos_lab. El script utlizado se puede ver al final del Apéndice A.
Resultado	Los dos servos se mueven a las posiciones y velocidades indicadas.
Requisito capacidad	RS.07
Estado	Prueba realizada con éxito

Tabla 5.9: Prueba de aceptación 7

PRU.A.08	Instalar Xenomai y comprobar su funcionamiento.
Descripción	Parchear el kernel de Linux con Xenomai para obtener un framework con restricciones de tiempo real.
Procedimiento	El procedimiento de instalación viene detallado en el apéndice B.
Resultado	La verificación viene comentada en el Apéndice B.
Requisito capacidad	RS.08
Estado	Prueba realizada con éxito

Tabla 5.10: Prueba de aceptación 8

En la figura 5.1 se muestra los numero de identifición de cada servo.

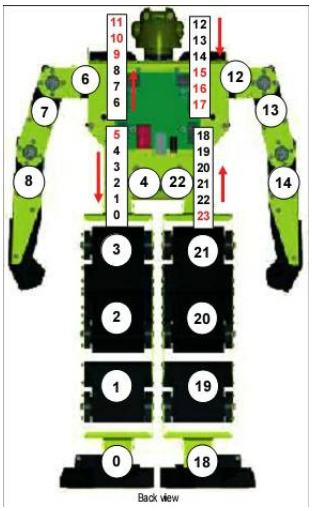

Figura 5.1: Idenfiticadores de servos en Robonova-I

5.6. Matriz de trazabilidad de funcionalidad

La matriz de trazabilidad se utiliza para saber que requerimientos quedan cubiertos por una prueba.

A continuación se presenta la tabla de requisitos. Se marcan con una X las intersecciones entre un requisito y las pruebas que permitan validarlo. Todos los requisitos quedan cubiertos por las pruebas.

	PRU.A.01	PRU.A.02	PRU.A.03	PRU.A.04	PRU.A.05	PRU.A.06	PRU.A.07	PRU.A.08
RS.01	X							
RS.02		X						
RS.03			X					
RS.04				X				
RS.05					X			
RS.06						X		
RS.07							X	
RS.08								X

Figura 5.2: Matriz de trazabilidad de funcionalidad

Capítulo 6

Planificación y presupuesto

6.1. Planificación

En esta sección se detalla el tiempo dedicado al desarrollo del trabajo. Se muestra el listado de tareas y el tiempo dedicado a cada una, acompañando el diagrama de Gantt para reflejarlo visualmente.

6.1.1. Listado de tareas

En la tabla 6.1 podemos ver las tareas realizadas, la fecha de inicio y de fin y la duración en días. No se tienen en cuenta los fines de semana ni festivos.

Tarea	Fecha de inicio	Fecha de fin	Duración (días)
Propuesta	14/09/12	14/09/12	1
Estudio Viabilidad	17/09/12	21/07/12	5
Análisis	9/01/13	23/01/13	11
Diseño	24/01/13	20/02/13	20
Implementación	21/02/13	30/04/13	42
Integración y pruebas	6/05/13	17/06/13	10
Documentación	20/05/13	20/06/13	24

Tabla 6.1: Planificación de las tareas

En la figura 6.1 se muestra el Diagrama de Gannt con las tareas desarrolladas.

Figura 6.1: Diagrama de Gantt

6.2. Presupuesto

En esta sección se detalla el presupuesto estimado para la realización del proyecto, haciendo un desglose por cada tipo (personal y hardware).

6.2.1. Tiempo dedicado

Una vez mostrados los días dedicados para el trabajo, procedemos a calcular las horas totales del mismo, estimando una media de dedicación de 6 horas por día.

- **Propuesta**: 1d x 2h = 2h.
- **Estudio de viabilidad**: 5d x 6h = 30h.
- **Análisis**: 11d x 6h = 66h.
- **Diseño**: 20d x 6h = 120h.
- **Implementación**: 42d x 6h = 252h.
- **Integración y pruebas**: 10d x 6h = 60h.
- **Documentación**: 24d x 6h = 144h.

Sumando todas las horas invertidas en cada tarea nos salen **un total de 674 horas dedicadas** en la ejecución de todo el trabajo.

6.2.2. Coste de personal

Una vez calculadas las horas, procedemos a calcular el coste de personal dedicado.

- Total días: 113
- Media de horas de trabajo diario: 6
- Dedicación hombre/mes (horas): 131,25
- Coste hombre/mes: 2.000 Euros

El coste de personal se calcula con la siguiete fórmula:

$$Coste\ de\ personal = \frac{Total\ días\ \ x\ \ Horas/día}{Dedicación\ \ hombre/mes} * Coste\ hombre/mes$$

Como resultado se obtiene la siguiente cantidad:

Apellidos y nombre	Categoría	Días trabajados	Horas al día	Coste (En euros)
Gallego Hernández, Fernando	Ingeniero	113	6	10.331,42

Tabla 6.2: Presupuesto de personal

6.2.3. Costes de hardware

Para la realización de este proyecto ha sido necesario la adquisición de los siguientes componentes informáticos:

- Monitor Monitor LG 22EA63V-P 21.5ÏPS LED (114,88)
- Ratón Logitech B110 Optical USB Mouse Negro OEM (4,87)

- Teclado B-Move Double Touch Kit Teclado + Ratón USB (5,52)

La tabla 6.3 de los componentes aquiridos para el proyecto.

Descripción	Coste sin IVA (Euros)	% uso dedicado al proyecto	Dedicación (meses)	Periodo de depreciación	Coste imputable
Monitor LG	114,88	100	6	60	11,48
Raton Logitech	4,87	100	6	60	0,49
Teclado B-Move	5,52	100	6	60	0,55
				TOTAL	**12,52**

Tabla 6.3: Amortización de componentes

Para el cálculo de la amortización se ha utilizado la siguiente fórmula:

$$Cálculo\ de\ la\ amortización = \frac{A}{B} * C * D$$

- A: nº de meses desde la fecha de facturación en que el equipo es utilizado.
- B: periodo de deprecación (6 meses)
- C: coste del equipo (sin IVA)
- D: % de uso que se dedica al proyecto

Para la realización del proyecto se ha necesitado la adquisición de los siguientes componentes:

- Placa Ebox 3310MX
- Servo Hitec hsr-8498hb
- Placa protoboard
- 2 conectores RS-232 DB-9 Hembra
- Soldador tipo lapiz 220V-30W

- 10 cables cortos
- Fuente de alimentación de 6V

La tabla 6.4 muestra los precios sin IVA de cada componente (en euros).

Concepto	Unidades	Coste unidad sin IVA	Coste total sin IVA
Placa Ebox	1	175,14	175,14
Servo Hitec	1	43,65	43,65
Placa protoboard	1	6,95	6,95
Conector RS-232	2	6,95	13,90
Soldador 220V-30W	1	9,90	9,90
Cable corto	10	0,10	1,00
Fuente	1	5,00	5,00

Tabla 6.4: Costes de Hardware

Los costes totales ascienden a 255,54 Euros.

6.2.4. Resumen de costes

La suma de cada tipo de coste se muestra la tabla 6.5.

Descripción de costes	Coste total
Personal	10.331,42
Amortización de equipos	12,52
Hardware	255,54
Costes indirectos(20 %)	2119,90
Total sin IVA	12.719,38
Total con IVA (21 %)	15.390,45

Tabla 6.5: Resumen de costes totales del trabajo

El presupuesto total es de QUINCE MIL TRESCIENTOS NOVENTA CON CUARENTA Y CINCO CÉNTIMOS DE EURO.

Capítulo 7

Conclusiones y trabajos futuros

Una vez expuestos las objetivos que nos han llevado a desarrollar el trabajo, expuesta la tecnología a utilizar y validado las pruebas efectuadas sobre los servos, vamos a realizar un resumen y conclusiones sobre este Trabajo de Fin de Grado y las posibilidades de mejorarlo y ampliarlo en el futuro.

7.1. Resumen

En esta sección vamos a sintetizar el trabajo realizado durante todo el proceso de desarrollo para mostrar una visión global.

Primero investigamos sobre el estado de los microcontroladores en robots comerciales para buscar alternativas a estos que permitieran utiliazar una librería de comunicación propia sobre un entorno de software libre como Linux.

El objetivo estaba en diseñar un sistema alternativo que consiguiera la misma funcionalidad. Una vez visto el objetivo, estudiamos los componentes para lograr su implantación.

Se propusieron como controladores dos placas diseñadas para aplicaciones roboticas como Ebox y Roboard, decantándonos por la primera por tener identicas características pero ser más barata.

Para la elección del firmware, se descartó desde un principio Windows y se opto Linux al tratarse de un sistema no propietario con el que programar

la librería en C.

Después se estudiaron los servos que forman parte del robot y la forma de comunicarse con ellos a través de comandos enviados por el puerto COM del controlador.

Con los componentes elegidos, el último paso fue estudiar la interconexión de los componentes y ensamblarlos para a continuación, poder realizar las pruebas de validación.

Con estos pasos seguidos se han llegado a las siguientes conclusiones.

7.2. Conclusiones del proyecto

- Para la realización el proyecto se han propuestos los componentes mas adecuados cumpliendo los sigiuentes requerimientos:
  - Bajo coste.
  - Reducido tamaño.
  - Utilizacion de software libre.
- Se consiguió realizar satisfactoriamente la conexión del servo con la placa para su control.
- Se codificó la librería en c y se creó un programa de prueba que enviara comandos al servo. Este respondió correctamente a todos los comandos.

Analizando lo anterior, podemos afirmar que el objetivo del trabajo se ha cumplido con éxito.

7.3. Conclusiones personales

La realización del proyecto me ha supuesto embarcarme en el mundo de la robótica, tocando temas de circuiterias que se salen un poco del temario de la carrera, pero que me han servido para ampliar mis conocimientos. Lo interesante del proyecto ha sido construir de cero el sistema y haber tenido que lidiar con las complicaciónes de conectar cada una de las partes.

La dificultad del proyecto ha estado en las conexiones de la placa con el servo, siendo lo más complicado del trabajo lograr que el servo recibiera y mandara algún dato. Una vez superado ese obstaculo, fue muy gratificante el conseguir que la librería desarrollada funcionase y ver el robot realizar los movimientos programados.

7.4. Trabajos futuros

Hemos comprobado que la placa Ebox cumple correctamente con los requisitos para la conexión con los servos del robot. Además, al disponer de puertos USB que posibilitan la conexión por bluetooth y WiFI podemos ampliar la funcionalidad para realizar un control en remoto.

Centrándonos en el desarrollo sofware, se puede modificar la librería para introducir funciones de tiempo real, consiguiendo movimientos más precisos en el tiempo. También a partir de movimietos simples, se pueden desarrollar macros que agrupen movimientos, creando subrutinas como por ejemplo, levantar brazo derecho. El trabajo se ha enfocado al control de los servos, pero en un futuro se puede ampliar a la comunicación con otros sensores del robot, como el de proximidad encargado de detectar obstáculos. Por último, se podría añadir a la librería funciones que utilicen algoritmos de inteligencia artificial para dotar al robot de movimientos autónomos.

Glosario

API: *Application Programming Interface* o Interfaz de programación de aplicaciones.

BIOS: *Basic Input-Output System* o Sistema Básico de Entrada/Salida.

CC: corriente continua o corriente directa.

Datasheet: es un documento que resume las características técnicas de un componente.

EEPROM: *Electrically-Erasable Programable Read-Only Memory* o ROM programable y borrada eléctricamente.

Firmware: bloque de instrucciones de máquina para propósitos específicos.

GCC: *GNU Compiler Collection* o Colección de Compiladores GNU.

GND: Tierra.

GNOME: *GNU Object Model Environment* o entorno de modelos orientado a objetos de GNU.

HMI: *Hitec Multi-protocol Interface* o interfaz multi protocolo de Hitec. LTS

LTS: *Long Term Support* o soporte de largo plazo.

PIC: son una familia de microcontroladores tipo RISC.

POSIX: *Portable Operating System Interface.*

Protoboard: tablero con orificios para prototipado de circuitos electrónicos.

PWM: *Pulse-duration modulation*,

RAM: *Random-access memory* o memmoria de acceso directo.

RXD: Recepción de datos.

TTL: *transistor-transistor logic*, tecnología de construcción de circuitos electrónicos digitales.

TXD: Envío de datos de transmisión.

USART: *Universal Synchronous Asynchronous Receiver Transmitter* .

VCC: Voltaje en corriente continua (en inglés VDD).

Bibliografía

[1] Gonzalo Zabala: Robótica, guía teórica y práctica.

[2] Antonio Barrientos: Fundamentos de Robótica. McGraw-Hill, 1997 . Poing, 2004.

[3] Martín Cuenca, E: Microcontroladores PIC. La solución en un chip. Paraninfo-ITP, 1998. Poing, 2004.

[4] Robert H. Bishop. The Mechatronics HandBook. CRC Press, 2002

[5] Pete Miles, Tom Carroll: Build Your Own Combat Robot. McGraw-Hill, 2002

[6] John J. Craig: Introducction to robotics. Mechanics and control. Longman, 1989

[7] Ebox 3310MX-AP Datasheet: `http://robosavvy.com/store/product_info.php/products_id/1854`

[8] Hitec HSR-8498hb Datasheet: `http://robosavvy.com/Builders/i-Bot/`

[9] Dynamixel AX-12 Datasheet: `http://www.electronickits.com/robot/BioloidAX-12(english).pdf`

[10] Kit Robonova I : `http://robosavvy.com/site/index.php?option=com_content&task=view&id=135`

[11] RS-232 vs TTL: `https://www.sparkfun.com/tutorials/215`

[12] Configurar el puerto COM con POSIX: `http://digilander.libero.it/robang/rubrica/serial.htm#3_1`

[13] Xenomai: Framework de Tiempo Real para Linux : http://www.xenomai.org/

[14] Historia de la robótica : http://robotiica.blogspot.com.es/2007/10/historia-de-la-robtica.html

[15] Sistemas operativos avanzados : http://www.arcos.inf.uc3m.es/~ssooaa/dokuwiki/lib/exe/fetch.php?id=programa&cache=cache&media=ssooaa_str_y_empotrados_v1.pdf.

[16] Wikipedia: RS-232: http://es.wikipedia.org/wiki/RS-232

[17] Wikipedia: USB: http://es.wikipedia.org/wiki/Universal_Serial_Bus

[18] Wikipedia: Servomotor: http://es.wikipedia.org/wiki/Servomotor

[19] Wikipedia: Linux : http://en.wikipedia.org/wiki/Linux

[20] Wikipedia: Puerto serie : http://es.wikipedia.org/wiki/Puerto_serie

Apéndice A: Manual de implantación y uso de la librería

Los ficheros creados para utilizar la librería son:

- **hsr-com.c**: Contiene el código fuente de la librería
- **hsr-com.h**: Fichero de atributos y cabeceras que se importará en el fichero .c.
- **hsr-com-script**: Programa binario que utiliza la librería para comunicarse con los servos.

Compilación:

Para compilar la librería se han ejecutado los siguientes comandos (se tiene que tener instalado el compilador GCC en Linux):

crear el fichero objeto .o

gcc -c hsr-com.c -o hsr-com.o

crear librería:

ar -rv libhsr-com.a hsr-com.o

enlazar la librería con programa de ejemplo (ver al final del apéndice):

gcc hsr-com-script.c -o hsr-com-script -L. -lhsr-com -static

Ejecución del programa:

Para ejecutar cada instrucción se han creado enlaces simbólicos al ejecutable con el nombre de cada uno de los comandos de la siguiete forma:

ln -s hsr-com-script [nombre del comando]

Para ejecutarlo, basta con escribir el nombre del comando seguido de los parámetros necesarios. Ej. move ttyS0.

Manual del programador para el uso de la librería

La interfaz que se describe a continuación permite acceder a cada metódo de instruccion de la librería desde una terminal de Linux y cuya estructrua es la siguiente:

Uso:

comando [opciones]

Opciones	Descripción
-p [puerto]	Permite seleccionar el puerto COM a utilizar (Ej: ttyS0). Sin flag, se coje por dejecto el puerto ttyS0.
-i [id]	Permite seleccionar el identificador del servo a utilizar.
-s [vel]	Permite seleccionar la velocidad de giro del servo.
-a [pos]	Permite seleccionar la posición en ángulos en la que se quiere que se mueva el servo.

A continuación se detallan cada uno de los comandos:

version

NOMBRE

version -p [puerto]

SINOPSIS

include "hsr.com.h"

DESCRIPCIÓN

El comando lee del servo la versión y el identificador del servo conectado mostrándolo por pantalla

ERRORES

COM_FAILURE: El puerto COM no está disponible.
COM_NO_RESPONSE: No se recibe señal del puerto serie.
ERROR_PARAMITER: El parámetro introducido no es correcto.

battery

NOMBRE

battery -p [puerto]

SINOPSIS

include "hsr.com.h"

DESCRIPCIÓN

El comando lee del servo el voltaje y amperaje del servo mostrándolo por pantalla

ERRORES

COM_FAILURE: El puerto COM no está disponible.
COM_NO_RESPONSE: No se recibe señal del puerto serie.
ERROR_PARAMITER: El parámetro introducido no es correcto.

speed

NOMBRE

speed -p [puerto] -i [identificador] -s [velocidad]

SINOPSIS

include "hsr.com.h"

DESCRIPCIÓN

El comando lee la posición actual del servo conectado y la muestra por pantalla

ERRORES

COM_FAILURE: El puerto COM no está disponible.
COM_NO_RESPONSE: No se recibe señal del puerto serie.
SPEED_OUT_OF_RANGE: Velocidad fuera de rango. Usar de 0 a 127.
ID_OUT_OF_RANGE: Identificador fuera de rango. Usar de 0 a 239.
ERROR_PARAMITER: El parámetro introducido no es correcto.
ERROR_NUMBERS PARAMITER: El numero de parámetros introducidos no es correcto.

position

NOMBRE

speed -p [puerto]

SINOPSIS

include ”hsr.com.h”

DESCRIPCIÓN

El comando establece la velocidad del servo indicado y lee su posición

ERRORES

COM_FAILURE: El puerto COM no está disponible.
COM_NO_RESPONSE: No se recibe señal del puerto serie.
SPEED_OUT_OF_RANGE: Velocidad fuera de rango. Usar de 0 a 255.
ID_OUT_OF_RANGE: Identificador fuera de rango. Usar de 0 a 239.
ERROR_PARAMITER: El parámetro introducido no es correcto.
ERROR_NUMBERS PARAMITER: El numero de parámetros introducidos no es correcto.

move

NOMBRE

move -p [puerto] -i[identificador] -a[posición]

SINOPSIS

include ”hsr.com.h”

DESCRIPCIÓN

El comando manda al servo la posición en ángulos a la que ha de moverse

ERRORES

COM_FAILURE: El puerto COM no está disponible.
COM_NO_RESPONSE: No se recibe señal del puerto serie.
ERROR_PARAMITER: El parámetro introducido no es correcto.
POSITION_OUT_OF_RANGE: Posición fuera de rango. Usar de -95 a 95 grados.
ERROR_NUMBERS PARAMITER: El numero de parámetros introducidos no es correcto.

moveall

NOMBRE

moveall -p [puerto]

SINOPSIS

include ”hsr.com.h”

DESCRIPCIÓN

Mueve todos los servos a una determinada posición

ERRORES

COM_FAILURE: El puerto COM no está disponible.
COM_NO_RESPONSE: No se recibe señal del puerto serie.
ERROR_PARAMITER: El parámetro introducido no es correcto.
ERROR_NUMBERS PARAMITER: El numero de parámetros introducidos no es correcto.

Con estos comandos se puede desarrolar un script, permitiendo realizar todo tipo de movimientos a diferentes velocidades y en el servo que se indíque en el comando.

Sheel Script

Listing 1: Script de prueba

```
./speed -p ttyS0 -i 7 -s 255
./speed -p ttyS0 -i 11 -s 255
./move -p ttyS0 -i 7 45
./move -p ttyS0 -i 11 45
./speed -p ttyS0 -i 7 -s 30
./speed -p ttyS0 -i 11 -s 30
./move -p ttyS0 -i 7 90
./move -p ttyS0 -i 11 90
./speed -p ttyS0 -i 7 -s 10
./speed -p ttyS0 -i 11 -s 10
./move -p ttyS0 -i 7 45
./move -p ttyS0 -i 11 45
./speed -p ttyS0 -i 7 -s 1
./speed -p ttyS0 -i 11 -s 1
./move -p ttyS0 -i 7 -s 0
./move -p ttyS0 -i 11 -s 0
```

Código fuente:

Listing 2: Fichero de cabeceras de la librería

```
//*********************************************
/*
```

```
* File: hsr-com.h
* Autor: Fernando Gallego HernÃ¡ndez
*
***********************************
*/

#ifndef HSR_H
#define HSR_H

#define HSR_MAX_SERVOS          18
#define HSR_BUFFER_IN_SIZE      7
#define HSR_BUFFER_OUT_SIZE     7

/**Default parameters**/
#define BAUDRATE        B19200

/** Instruction Set **/
#define B_VERSION       0xE7
#define B_BATTERY       0xE8
#define B_Read_ADC_pos      0xE5
#define B_ID_R_POS_PC     0xE9
#define B_ID_W_MOV_MAX        0
#define B_motor_go_stop     0xEB
#define B_Write_CMD       0xE6


/** Key Instruction Set **/
#define K_VERSION       1
#define K_BATTERY       2
#define K_Read_ADC_pos      3
#define K_ID_R_POS_PC     4
#define K_ID_W_MOV_MAX      5
#define K_Write_CMD       6

#define C_COM_PORT        6
#define C_BAUDRATE        7



typedef unsigned char byte;

void hsrInit(long baud);

/** Commands HSR-9849: **/
int CommResource(char *argv);
int gekeyCommand(char *command);
int getIDVersion(int argc);
int getCurrentVoltage(int argc);
int getPosition(int argc);
```

```
int setMotorSpeedReadPosition(int argc, char *argv[]);
int setPosition(int argc, char *argv[]);
int setAllPosition(int argc, char *argv[]);
int sendPacket(int h, unsigned char *buf, int size);
int receivingPacket();
char RxTx_chk(char dt0, char dt1, char dt2, char dt3);


#endif
```

Código fuente del programa de ejemplo:

Listing 3: Fichero fuente del programa de ejemplo

```
//*************************************************
/*
* File: hsr-com.c
* Autor: Fernando Gallego HernÃ¡ndez
*
*************************************************
*/

#include              /*Library header file*/
#include <stdio.h>
#include <string.h> /* String function definitions */


/**********************
 * main
 **********************/

int main(int argc, char *argv[]) {

  char buf[16] = { 0 };
  char command[16];
  int key;
  int  nCmds, cmd = 0, written, readB = 0, condition = 1;

  if (argc < 2) {
    printf(                          );
    return -1;
  }
  if(strlen(argv[1])>16){
    printf(                                          );
    return -1;
  }
  if (CommResource(argv[1]) == -1){
    return -1;
  }
```

```
    else{
      printf(                );
    }

    // switch to treat each command.
    strcpy(command, argv[0]);

    //obtein the index
    key = gekeyCommand(command);

    switch (key) {
      case K_VERSION: getIDVersion(argc);
        break;
      case K_BATTERY: getCurrentVoltage(argc);
        break;
      case K_Read_ADC_pos: getPosition(argc);
        break;
      case K_ID_R_POS_PC: setMotorSpeedReadPosition(argc, argv);
        break;
      case K_ID_W_MOV_MAX: setPosition(argc, argv);
        break;
      case K_Write_CMD: setAllPosition(argc, argv);
        break;
      default: printf(                        );
        return 0;
        break;
    }
    return 0;
  }
```

Apéndice B: Instalación del entorno

En este apéndice vamos a detallar la instalación de Lubuntu en la placa Ebox, parcheando el kernel para obtener capacidades de tiempo real con Xenomai. Al final se cuenta como habilidar el puerto COM una vez instalado todo el sistema.

.1. Instalación de Lubuntu

Los pasos llevados a cabo son los siguientes:

1. Descargar e instalar el programa VirtualBox en un ordenador.
2. Abrir VirtualBox y pulsar en crear nueva máquina virtual.
3. Se selecciona una memoria de 256 MB.
4. No seleccionar bootear sobre disco duro.
5. Finalizar.
6. Conectamos la tarjeta SD al ordenador con un adaptador USB. Ejecutamos el administrador de disco para apuntar el *USB device number* que necesitaremos a continuación.
7. Abrir una terminal con el comando cmd (se requiere hacerlo en modo administrador).
8. Cambiar al directorio donde tengamos instalado virtual box, en mi caso: C:/Program Files/Oracle/Virtualbox.

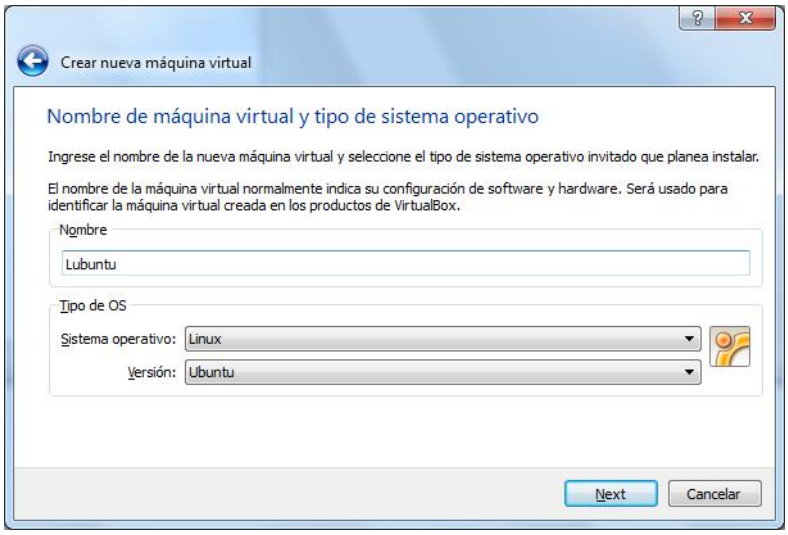

Figura 1: Virtual box: Crear nueva máquina virtual

9. Ejecutar la siguiente instrucción: vboxmanage internalcommands createrawvmdk -filename [ruta absoluta del fichero de salida] -rawdisk [Especificador de disco] -register En mi caso:

```
c:\Program Files\Oracle\VirtualBox>vboxmanage internalcommands createrawvmdk -fi
lename c:\users\hung-wen\.virtualbox\usb.vmdk -rawdisk \\.\physicaldrive1 -regis
ter
Oracle VM VirtualBox Command Line Management Interface Version 3.2.6
(C) 2005-2010 Oracle Corporation
All rights reserved.

RAW host disk access VMDK file c:\users\hung-wen\.virtualbox\usb.vmdk created su
ccessfully.
```

Figura 2: cmd: Creación de archivo wvmdk.

10. Lanzamos nuevamente Virtualbox en modo administrador.
11. Seleccionamos la máquina virtual Lubuntu creada.
12. Pulsamos en almacenamiento.
13. En el Controlador IDE buscamos la ruta de la imagen de Lubuntu y la seleccionamos. En el atributo slot seleccionamos IDE Master Primario.
14. A continuación montamos el archivo usb.vmdk creado en el paso 10. En el atributo slot seleccionamos IDE Esclavo Primario.
15. Pulsamos sobre la maquina virtual y damos a empezar.

16. A continuación comienza la instalación de Lubuntu. Hay una serie de recomendaciones:

    Elegir manualmente las particiones:

    - Crear una partición principal para el sistema operativo (con el punto de montaje /) y sistema de fichero ext3.
    - Crear una partición de 750Mb para la memoria SWAP.

17. Seguir la instalacion estandar (elección de idioma, teclado, nombre del equipo...) y esperar para finalizar.

Paso 1: Elección de almacenamiento externo.

Se elije instalar Linux en una tarjeta SD. Para lograr buen rendimiento y fiabilidad, la tarjeta proviene de un vendedor de prestigio como Sandisk y además es de clase 10, que garantiza rapidez en las lecturas y escrituras.(El modelo es SanDisk MicroSD 8GB Ultra Clase 10). Un disco duro conectado a la placa es otra opción pero es menos transportable para el trabajo que requerimos. Una tarjeta SD es mas pequeña y permite cambiar fácilmente de sistema operativo, con la posibilidad de tener varios para ir desarrollando nuestras pruebas.

Paso 2: Preparación.

Descargamos la imagen Lubuntu 10.04 LTS de la web oficial.

Paso 3: Instalación del Sistema Operativo.

Para instalar Lubuntu en la SD se utilizó la máquina virtual VirtualBox. Pasos a seguir:

Paso 4: Primer arranque. Al arrancar en la maquina virtual vemos que se demora unos minutos, que no funciona del todo fluido y que el sistema trae aplicaciones que no necesitamos. Para lograr mayor rendimiento, podemos desinstalar aplicaciones innecesarias (procesador de texto, juegos) con el gestor de paquetes Synaptic e instalar un nuevo kernel que se adapte mejor a nuestro procesador y que contenga sólo los drivers que necesitamos (ethernet, USB y COM). A continuación se explica como instalar el nuevo kernel.

Paso 5: Compilación del kernel. En la máquina virtual hacemos lo siguiente:

1. Descargargando y desempaquetando el kernel. Descargamos el paquete de kernel de linux 2.6.38.8 de la página oficial roboard, el cual viene ya configurado para adaptarse a nuestra placa [1]. Lo descomprimimos, vamos a la carpeta boot y extraemos el fichero de configuración config-2.6.34.10-vortex86-sg. Descargamos las fuentes de la página de kernel.org [2] y las extraemos en una carpeta en el escritorio (ej. linux-kernel). A continuación, copiamos el fichero de configuración en la carpeta en raiz.

2. Configurando el kernel Abrimos a una terminal y nos posicionamos en la carpeta con el comando cd. (ej: cd /home/roboard/desktop/linux-kernel. Ejecutamos el comando make menuconfig. NOTA: Necesitas tener instalado las las librerías *ncurses*: Para instalarlas ejecutar el comando: sudo apt-get install ncurses-dev. Aquí se activarán los módulos que permitan reconocer nuestra tarjeta SD y puertos USB. Con la placa realizaremos la instalación del sistema operativo, la programación de la librería y realizaremos las pruebas necesarias para comprobar la comunicación con los servos.

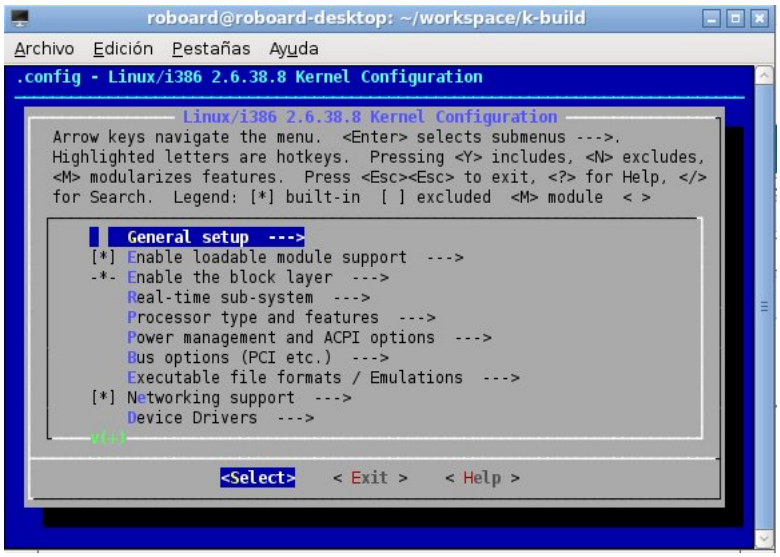

Figura 3: Menu config

3. Compilando Terminada la configuración del kernel obtenemos un fichero .deb que posteriormente instalaremos.

[1] http://www.roboard.com/
[2] https://www.kernel.org/pub/linux/kernel/v2.6/

Paso 6: Instalación de Xenomai. Según la definición que se hace en su propia pagina web, Xenomai es ün entorno de desarrollo en tiempo real que coopera con el núcleo Linux para proporcionar un respaldo omnipresente, independiente de la interfaz y muy cercano al tiempo real a aplicaciones de espacio de usuario, integrado completamente en el entorno GNU/Linux.

De forma sencilla, podemos decir que Xenomai proporciona funciones, estructuras y tipos de datos para desarrollar aplicaciones con restricciones de tiempo real, que se integran perfectamente con un entorno de Linux.

La instalación de Xenomai consta de dos partes:

1. Parche para el kernel de Linux Vamos a la página oficial de xenomai y descargamos la última versión de las fuentes (en mi caso v2.6) [3]. La descomprimimos en el escritorio, en una carpeta que llamamos xenomai. Abrimos el fichero README.INSTALL que se encuentra dentro de la carpeta descomprimida y seguimos las instrucciones de instalación para la arquitectura x86. Los pasos son los siguientes: Ejecutamos el script prepare-kernel.sh ejecutando el siguiente comando: cd xenomai scripts/prepare-kernel.sh –linux=../linux-kernel

   –adeos=ksrc/arch/x86/patches/adeos-ipipe-2.6.38.8-x86-2.11-01.patch –arch=i586.

Paso 7: Instalación y arranque con el nuevo kernel. Introducimos la tarjeta SD en la placa, y conectamos monitor, ratón y teclado. Para testear la instalación de xenomai, probamos a iniciar el kernel seleccionandolo en el menú de arranque. En el log de arranque debe de aparecer mensajes como: I-pipe: head domain Xenomai registered. Xenomai: hal/¡arch¿started. Xenomai: scheduling class idle registered. Xenomai: scheduling class rt registered. Xenomai: real-time nucleus v2.6.1 (Light Years Away) loaded. Xenomai: debug mode enabled. Xenomai: starting native API services. Xenomai: starting POSIX services. Xenomai: starting RTDM services. Para testear el soporte a usuario de Xenomai, lanzamos un test de latencia con el comando xeno latency. El test de latencia debe mostrar un mensaje cada segundo, con valores de mínimo, máximo y latencia como los siguientes:

[3] https://www.xenomai.org/

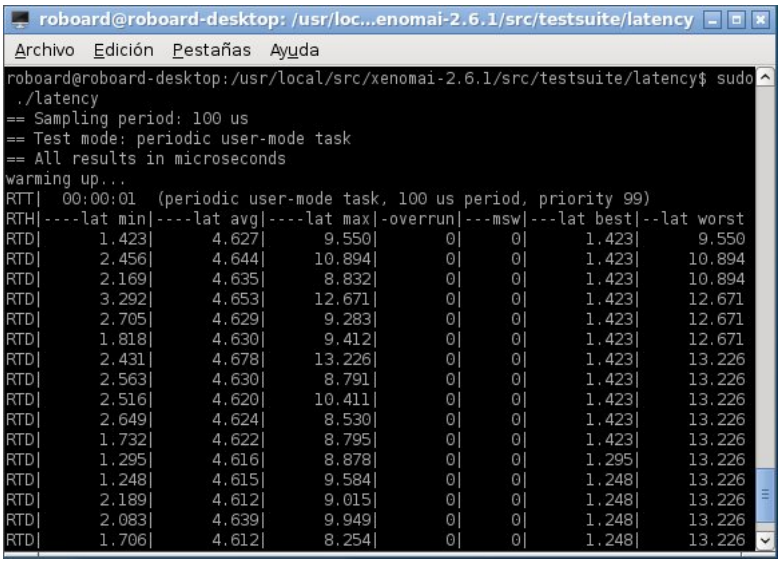

Figura 4: Respuesta al ejecutar el comando latency.

.2. Habilitación del puerto COM

Abrir una terminal y escribir: sudo apt-get install setserial

A continuación escribir: setserial /dev/ttyS2 irq 10 port 0x03e8 uart 16550A

Nota: El numero irq y dirección debe coincidir con la configuración de en la BIOS de la placa. Para verla, al reinicial pulsar F10 para entrar en la BIOS e irse al menu Chipset.

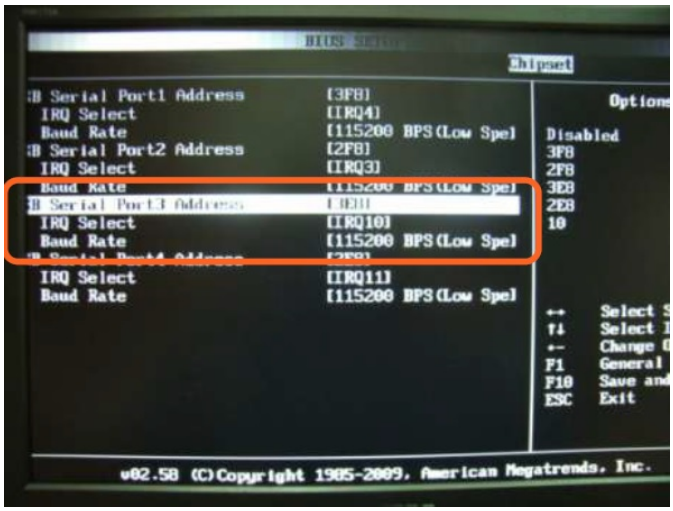

Figura 5: Respuesta al ejecutar el comando latency.

Con esto realizado, se pueden ver todos los puertos series instalados escribiendo el siguiente comando:

setserial -g /dev/ttyS*